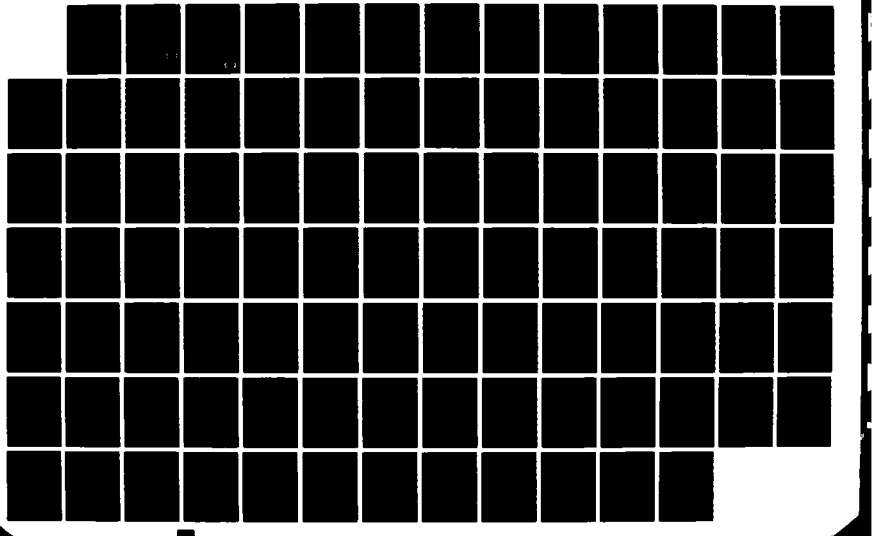
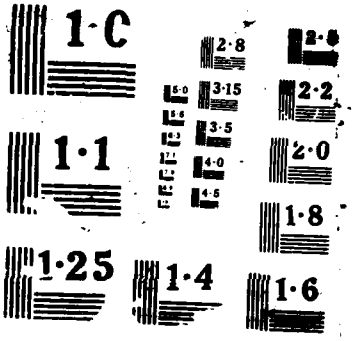
AD-A189 746 HIGH SPEED TRANSCENDENTAL ELEMENTARY FUNCTION ARCHITECTURE IN SUPPORT OF T..(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. 1/1

UNCLASSIFIED M J BAILEY DEC 87 AFIT/GE/ENG/87D-3 F/G 12/1 NL




1·C
1·1
1·25
1·4
2·8
3·15
3·5
4·0
4·5
2·2
2·0
1·8
1·6

DTIC FILE COPY

AD-A189 746

DTIC
ELECTE
MAR 0 2 1988
S D
H

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

88 3 01 121

AFIT/GE/ENG/87D-3

HIGH SPEED TRANSCENDENTAL ELEMENTARY FUNCTION ARCHITECTURE IN SUPPORT OF THE VECTOR WAVE EQUATION (VWE)

THESIS

Mickey J. Bailey
Captain, USAF

AFIT/GE/ENG/87D-3

Approved for public release; distribution unlimited

DTIC
ELECTE
MAR 0 2 1988
S D
H

AFIT/GE/ENG/87D-3

HIGH SPEED TRANSCENDENTAL ELEMENTARY FUNCTION ARCHITECTURE IN SUPPORT OF THE VECTOR WAVE EQUATION (VWE)

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Electrical Engineering

Mickey J. Bailey, B.S.
Captain, USAF

December 1987

Approved for public release; distribution unlimited

Preface

This research is part of the continuing development effort from the Vector Wave Equation (VWE) Research Group. The initial effort of this thesis was to find high speed architectures for the sine, cosine, and reciprocal functions required to implement the VWE processor.

One of the methods found for computing sine and cosine led me in a direction different from the original intent of the research. The method was the expansion in Chebyshev polynomials. The convergence of the Chebyshev polynomials is very good, requiring very few iterations to approximate the required functions. The hardware to compute the Chebyshev polynomials can be pipelined and would be very fast. The research at this point turned to developing a Chebyshev processor for the high speed computation of certain transcendental elementary functions.

I would like to thank my thesis advisor, Major J. DeGroat for his help during this thesis effort. I would also like to thank my wife and sons for their support throughout the Master's Degree program.

Mickey J. Bailey

Accession For
NTIS GRA&I
DTIC TAB
Unannounced
Justification
By
Distribution/
Availability Codes
Dist | Avail and/or Special
A-1

Table of Contents

List of Figures

List of Tables

Abstract

In support of a Very High Speed Integrated Circuit (VHSIC) class processor for computation of a set of equations known as the Vector Wave Equations (VWE), certain elementary fuctions including sine, cosine, and division are required. These elementary functions are the bottlenecks in the VWE processor. Floating point multipliers and adders comprise the remainder of the pipeline stages in the VWE processor.

To speed up the computation of the elementary functions, pipelining within the functions is considered. To compute sine, cosine, and division, the CORDIC algorithm is presented. Another method for computation of sine and cosine is the expansion of the Chebyshev polynominals.

The equations for the CORDIC processor are recursive and the resulting hardware is very simple, consisting of three adders, three shifters, and lookup table for some of the coefficients. The shifters replace the multiplies, because in binary, i right shifts is the same as multiplying by 2^i.

The expansion of the Chebyshev polynominals can be used to compute other trigonometric functions as well as the exponential and logarithmic functions. The expansion of the Chebyshev polynominals can be used as a mathematic coprocessor. From these equations, a pipelined architecture can be realized that results in very fast computation times. The transformation of these equations as a function of x instead of the Chebyshev polynominals produces an architecture which requires less hardware, resulting in even faster computation times.

HIGH SPEED TRANSCENDENTAL ELEMENTARY FUNCTION ARCHITECTURE IN SUPPORT OF THE VECTOR WAVE EQUATTION(VWE)

I. Introduction

This thesis is the study, development, and analysis of algorithms to compute transcendental elementary functions. The elementary fuctions presented in this thesis include trigonometric, exponential, logarithmic, and division functions. The hardware and computation times for each of the functions is also presented.

Three of the elementary functions - sine, cosine, and division - are needed to solve a set of equations known as the Vector Wave Equations (VWE). These equations are the basis for the VWE Processor, an extremely fast, highly pipelined architecture for computing the VWE. The bottlenecks in the VWE processor are the elementary functions listed above, as very fast floating point multipliers and adders comprise the remainder of the pipeline stages in the VWE processor.

To speed up the computation of the elementary functions, pipelining within the functions is considered. Two methods for computing some of the elementary functions are CORDIC and expansions of the functions as Chebyshev polynominals. Both of these methods are presented in this thesis.

Chebyshev polynominals can be used to compute functions other than those required by the Vector Wave Equation. These functions include other trigonometric functions as well as

the exponential and logarithmic functions. The expansion of the Chebyshev polynominals can be translated into an architecture for a high speed processor. From these equations, a pipelined architecture can be realized that results in very fast computation times. Manipulation of these equations produces an architecture which requires less hardware, resulting in even faster computation times. The development of a Chebyshev Processor is also presented in this thesis.

Vector Wave Equation Background

In 1984, a thesis topic to find an algorithm which would solve Maxwell's equations for finding the magnetic (H) and electric (E) fields and vector potential (A) from an antenna (current source) was presented. This algorithm would provide the specification for a hardware design. This design would be optimized to compute the fields and potential in as little time as possible. Classical methods of computing these fields and vector potential are so time comsuming that only very simple antenna geometries are considered. These equations require millions of floating point operations, thus resulting in a N-P Complete Problem: a problem in which the solution is needed before the computer has time to solve it. Jones (11), Hoyt (7), and Strauss (14) have found solutions to the equations which could result in tremendous computational savings (orders of magnitude) by proper manipulation of the VWE.

From basic physics one knows that the effect of radar on an aircraft is that of producing a radar cross section (RCS). A transmitting antenna produces an incident wave on the aircraft. The RCS is defined as the "area intercepting that amount of power which, when scattered isotropically, produces at the receiver a density which is equal to that scattered by the actual target (1:65)." The goal was to provide algorithms which could be implemented in

hardware such that an interactive CAD/CAM system could be used to determine the effect a design would have on the radar cross section. One could determine the fields and vector potential in the design phase instead of finding out in the test phase that the airplane's RCS is unacceptable. At this point, either modifications will have to be made to the aircraft or some type of absorptive material will have to be applied to the structure.

Jones' contribution to the VWE project was to find an algorithm for computing the vector potential, **A**, by solving the radiation integral as approximated by a mid-point summation (11:6,34). For his thesis, Jones used a simple dipole antenna orientated along the z-axis. That is, for a simple dipole, the vector potential is accurate to 2 decimal places when the dipole is broken into 500 sub-elements for the summation.

Hoyt was faced with the problem of soving the **E** and **H** fields. Hoyt used the same mid-point summation technique used by Jones to obtain algorithms for these fields using the same simple dipole along the z-axis (7:7). Algorithms were produced which compute both the real and imaginary parts of the *x*, *y*, and *z* components of the vector potential, electric, and magnetic fields. A total of 18 summations, each requiring 500 subelements are required. Jones found that for M equal to 500 the result from the mid-point summation technique was accurate to 2 decimal places. His results were also accurate to 2 decimal places for M equal to 500.

Now that Jones and Hoyt had provided algorithms for computing **A**, **E**, and **H**, Strauss was able to see that many of the intermediate results from computing **A**, **E**, and **H**, if saved, could be later used in the cartesian coordinte solution of these equations (14). Each field thus consists of an *x*, *y*, and *z* direction in the cartesian coordinate system. Strauss provided a data flow chart which is what he termed the "parallel VWE algorithm" (14). He demonstrated that "a total of 12 computational levels are needed if the architecuture supports 30 floating point

multiplies, 18 floating point additions/subtractions, a square root, sin/cos, and an inverse or division operation" to produce the 18 components of **A**, **E**, and **H** (14:42).

The data flow chart could be used to implement the VWE in either software or hardware. The software approach would be to optimize the parallelism inherent in the data flow chart. This approach was undertaken by Dave Allen for his thesis. The hardware implementation would result in a pipeline, since Strauss had in effect found the top level pipeline to solve the VWE. Pipelining is achieved by "subdividing the input task into a sequence of subtasks, each of which can be executed by a specialized hardware stage that operates concurrently with other stages in the pipeline. Successive tasks are streamed into the pipe and get executed in an overlapped fashion at the subtask level" (8:145).

For the hardware implementation, the top level pipeline would need to be broken down into component pipelines. This is necessary because the slowest element in the pipeline sets the speed for the rest of the pipeline. Pipelining of the multipliers, adders, and subtractors was not necessary. However, the computation of the sine, cosine, square root, and division would each require their own internal pipeline so as not to slow down the top level pipeline.

Chebyshev Coprocessor Background

One method considered to compute the sine and cosine was the expansion of these functions as Chebyshev polynominals. Chebyshev polynominals meet the minmax property and converge more quickly than other polynominal expansions (5:2). The Chebyshev series also has "the same convergence properties as those of the Fourier series, but generally with a much faster rate of convergence" (5:31). The Chebyshev polynominals are recursive, and would therefore appear to be prime candidates for a pipeline approach, like that of the VWE.

Apparently, very little progress has been done in this area for evaluating elementary functions since 1970, with the exception of (9:128).

The equations for the expansion in Chebyshev polynomials for the sine and cosine, when expanded, can be rearranged in a form which resembles a power series expansion. New equations were found that, with a change of constants, were in terms of powers of x instead of the Chebyshev polynomials. The hardware implementation of these equations requires less hardware per pipeline stage than implementation of the equations in terms of the Chebyshev polynominals. This began a search for representation of other elementary functions in Chebyshev polynominals that could also be expressed in terms of x. The result would be a pipelined Chebyshev processor for elementary functions such as sine, cosine, tangent, arctangent, natural logarithm, and exponential.

Problem

The basic problem is twofold: one, develop fast hardware for the elementary functions required by the VWE Processor; and second, develop the hardware for the Chebyshev Processor. Hardware for range reduction and scaling of the inputs for these functions must also be determined.

Scope

The scope of this effort is to design hardware for the elementary functions for use in the VWE Processor and the Chebyshev Processor, including range reduction and scaling of the inputs to the processors as required. An estimate of the computation times for the functions will be evaluated. The elementary functions for the VWE Processor are limited to sine,

cosine, and division. The elementary functions for the Chebyshev Processor are limited to the sine, cosine, tangent, arctangent, natural logarithm, and exponential. The hardware presented in this thesis is limited to single point precision as defined in the IEEE Standard 754 (10:3).

Organization

The remainder of this thesis is organized as follows. In Chapter II, the elementary functions required for the VWE processor are presented. Range reduction formulas and scaling are discussed as required. Chapter III is the developement of the Chebyshev processor for the trigonometric, exponential, and logarithm fuctions. Chapter III includes the manipulation of the Chebyshev polynomials for hardware reduction. Chapter IV is an analysis of the results in terms of timing for the hardware presented in the previous two chapters. The hardware for the Chebyshev processor is also presented. And Chapter V is the conslusion of the thesis and also contains recommendations for follow-on work.

II. Elementary Functions for VWE Processor

As stated in Chapter I, hardware support is required to calculate the sine, cosine, and division of intermediate results in the course of solving the VWE. In this chapter, the algorithms and hardware for solving these functions will be presented. The CORDIC method for solving sine, cosine, and division is presented first, followed by the Chebyshev method for computing sine and cosine. For both the CORDIC and Chebyshev methods, preprocessing, which includes range reduction of the argument, and postprocessing are discussed and the hardware for the each of the methods is presented.

CORDIC Algorithm

The Coordinate Rotational Digital Computer (CORDIC) was introduced by (15:330-334). CORDIC was used to solve "the trigonometric relationships involved in plane coordinate rotation and conversion from rectangular to polar coordinates" (15:330). The CORDIC algorithm can be used to solve not only the elementary functions such as the trigometric and logarithmic functions, but they can also be used for multiplication, division, and conversion from decimal to binary (3:335-339).

The iterative equations that CORDIC uses to solve for the various functions are as follows (13:1283):

$$Z_{i+1} = Z_i - d_i e_i \tag{2.1}$$

$$X_{i+1} = X_i - m d_i Y_i 2^{-i} \tag{2.2}$$

$$Y_{i+1} = Y_i + d_i X_i 2^{-i} \tag{2.3}$$

where X_0, Y_0, and Z_0 are inputs, and m, d_i, and e_i depend on the function to be calculated.

The final answer depends on the function being caluclated, as the final result may be any combination of Z_i, X_i, or Y_i. The initial values of Z, X, and Y also depend on the function being evaluated.

The effect of the equations is to rotate the input through a given sequence of angles until the angle of rotation converges to zero. The input is given in terms of the magnitude of the vector it represents in the Cartesian coordinate system. The input is rotated in the following sequence of degrees: +/- 90, +/- 45, +/- 26.5 and so on, to converge on a rotation of 0 degrees. The X and Y components of the input argument are computed for each rotation and are added or subtracted from the previous rotation. The unique selection of the angles of rotation is such that the rotation of the components can be done by shifting and adding, as seen in Equations (2.2) and (2.3). The shifting is done in place of the multiplication because a multiply by 2^{-1} is the same as i unitary right shifts in binary.

To calculate the sine and cosine of the argument, α, the correct assignments are made to Equations (2.1) - (2.3). The argument, α must be range normalized as discussed in the next section. The CORDIC equations to calculate sine and cosine are shown in Equations (2.4) - (2.6).

$$Z_{i+1} = Z_i - sign(Z_i)(tan^{-1}(2^{-i})) \tag{2.4}$$

$$X_{i+1} = X_i - sign(Z_i)\,(2^{-i})\,(Y_i) \tag{2.5}$$

$$Y_{i+1} = Y_i + sign(Z_i)\,(2^{-i})\,(X_i) \tag{2.6}$$

where

Z_0 = range normalized α
$X_0 = 1$
$Y_0 = 0$

After i iterations, X_{i+1} is proportional to the cos (Z_0) and Y_{i+1} is proportional to the sin (Z_0) (13:1283). The results are proportional to the sine and cosine because an error factor of $1/K$ is introduced in each step of the iteration. In order to compensate for this, X_0 is set equal to K, where

$$K = \prod_{i=0}^{\infty} \cos(tan^{-1}(2^{-i})) \tag{2.7}$$

and Y_0 is set equal to 0.

One benefit of these equations is that they solve for the both the *sine* (Z_0) and *cosine* (Z_0) simultaneously. Other benefits are realized in the hardware and are discussed in the hardware section.

Each iteration of Equations (2.3) - (2.6) results in one additional bit of accuracy for the sine and cosine of Z_0. To meet the IEEE Standard 754 for single precision (10:3), 24 iterations will be required. This results in an accuracy of 2^{-24} or $6 * 10^{-08}$, which is 7 to 8 decimal places of accuracy. For 24 iterations, $K = 0.607252936$.

Preprocessing and Range Reduction. The argument, α for the CORDIC algorithm must be range normalized to the range $[-\pi/2, \pi/2]$. Range reduction for sine and cosine is possible because of the periodicity of the functions. Additionally, the positive argument is folded into the first quadrant and the negative argument is folded into the fourth quadrant. One equation to reduce the argument to this range is found in (2:107). The equation is

$$\alpha = (a * \pi) + b \tag{2.8}$$

where b is in the range $[-\pi/2, \pi/2]$.

To find a in Equation (2.8), divide both sides by π:

$$a = (\alpha/\pi) - (b/\pi) \tag{2.9}$$

where (b/π) is in the interval $[-1/2, 1/2]$

Define

$$N = \textit{Integer portion of } (\alpha/\pi) \quad \textit{or} \quad N = \lfloor (\alpha/\pi) \rfloor \tag{2.10}$$

and

$$F = \textit{Fractional portion of } (\alpha/\pi) \quad \textit{or} \quad F = (\alpha/\pi) - \lfloor (\alpha/\pi) \rfloor \tag{2.11}$$

where $0 \leq F < 1$.

The effect of multiplying α by $1/\pi$ is to find a multiple of π such that the integer part of α/π, N, is a multiple of π, and the fractional part, F, is a rotation or offset, in radians, from either 0 or π. If N is even, then $\pi*F$ is the offset from 0 or the positive x axis (Figure 2.1). If N is odd, then $F*\pi$ is the offset from π or the negative x axis. The direction of rotation from the axis depends on the sign of N. If N is positive, then the rotation is counterclockwise. If N is negative, then the rotation is clockwise.

The input to the CORDIC processor, Z_0, is equal to $F * \pi$. But, as seen in Equation (2.11), the range of F is too large. F must be further reduced to the range [0, 0.5]. This is

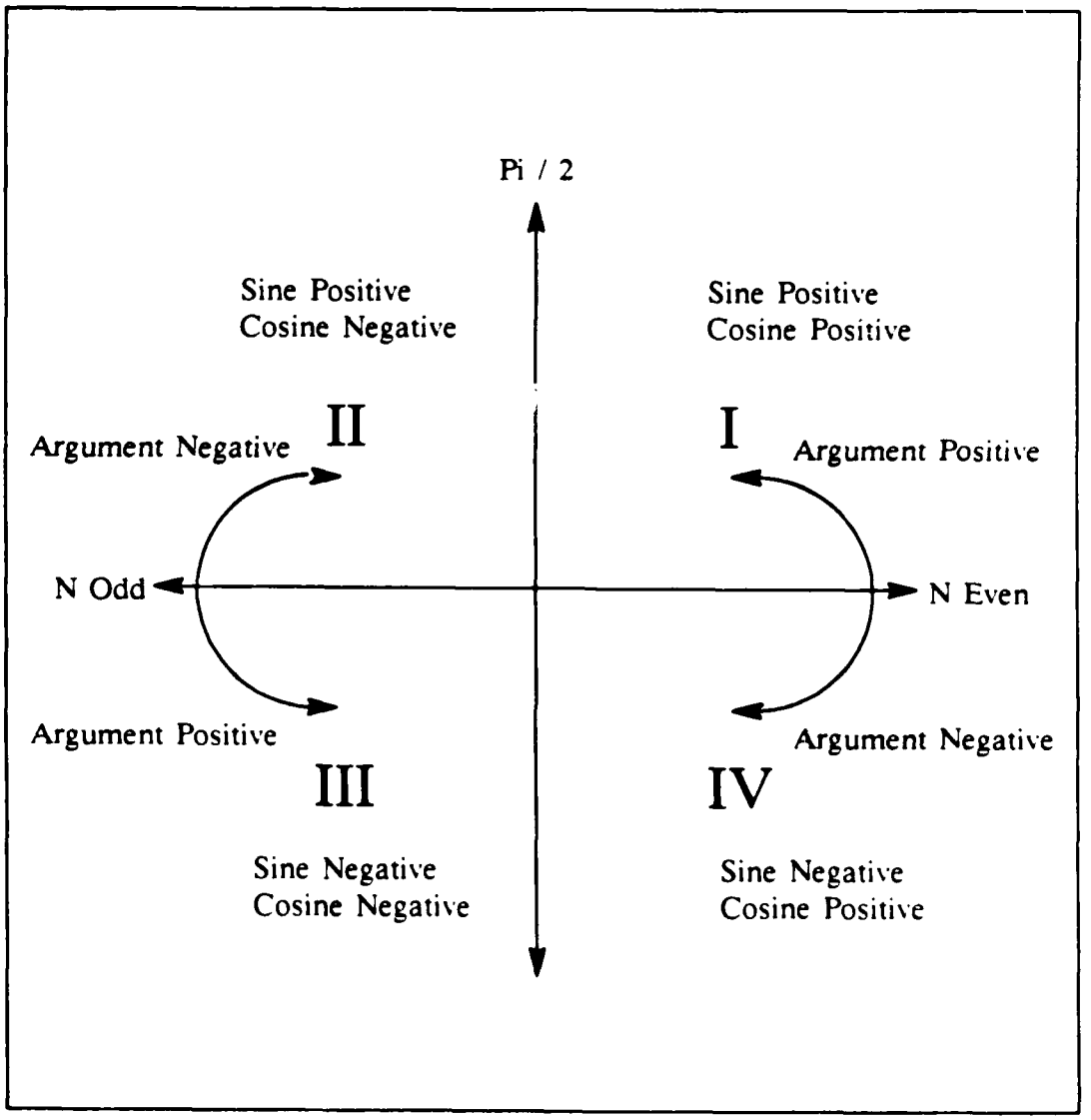


Figure 2.1. Sign of Sine and Cosine

accomplished by subtracting 0.5 from F if $F \geq 0.5$. But if 0.5 is subtracted from F, then 0.5 must be added to N so that the equality $(\alpha/\pi) = N + F$ will be maintained.

Consider the case where α is positive. After the argument is multiplied by $1/\pi$, the argument is folded into either the first or second quadrants if N is even, the third or fourth quadrants if N is odd. Then F is in the range [0, 1). However, F must be in the range [0, 1/2]. To accomplish this, if $F \geq 1/2$, then N is incremented. To maintain the equality, 1/2 is subtracted from F. The argument is now folded into either the first or fourth quadrant. Finally, F is multiplied by π to satisfy Equation (2.8). The input to the CORDIC processor is $(F^*\pi)$.

Basically the same applies when α is negative except both N and F are negative. The difference is that after α is multiplied by $1/\pi$, if $F \geq -1/2$, then -1 is added to N. Again, to maintain the equality, $-1/2$ is subtracted from F. The input to the CORDIC processor is still $(F^*\pi)$.

Fortunately, the CORDIC equations for the sine and cosine compensate for the sign of the final answer if the argument, α, is positive. Thus, X_{24} and Y_{24} have the correct sign if α is positive. However, if α is negative, then both X_{24} and Y_{24} must be multiplied by -1 to obtain the correct sign.

The input to the CORDIC Processor for the VWE Processor is in the IEEE Standard 754 floating point format (10:3). As will be seen in the hardware section, the hardware for the CORDIC Processor is fixed point. Therefore, part of the range reduction includes a conversion from floating point to fixed point.

The following steps are necessary for range normalization of α.

(1) If α is positive, the *sign* equals + 1.

Otherwise, the *sign* equals -1.

(2) Compute $\beta = \alpha(1/\pi)$, which consists of an integer part, N, and a fraction part, F.

(3) If $|F| \geq 0.5$, then $F = F-(sign * 0.5)$ and $N = N+(sign * 1)$.

(4) Compute $Z_0 = F*\pi$.

(5) Convert Z_0 to fixed point and loaded into the Z register of the CORDIC processor.

CORDIC Postprocessor. As stated previously, the sign of X_{24} and Y_{24} must be corrected. Also, results from the CORDIC hardware are in fixed point format, so normalization and conversion to the floating point format must be done. These functions must be accomplished in the postprocessor. The inputs to the postprocessor are the *sign*, X_{24} and Y_{24} where X_{24} and Y_{24} are the final values output from the CORDIC hardware.

The postprocessing consists of the following:

(1) Normalize X_{24} and Y_{24}.

(2) Convert X_{24} and Y_{24} to the floating point format.

(3) If *sign* equals -1, invert the first bit of the floating point numbers, X_{24} and Y_{24}, as the first bit of the floating point number is the sign bit.

CORDIC Hardware. The benefits of using the CORDIC equations in a binary machine are realized in the hardware as the hardware for the CORDIC processor is quite simple, consisting of three adders/subtractors and three shifters, plus interconnection (Figure 2.2). The adders/subtractors can be fixed point. This results in faster computation times as fixed point adders and subtractors are faster than floating point adders and subtractors. One reason is that the floating point adders and subtractors must perform a normalization after

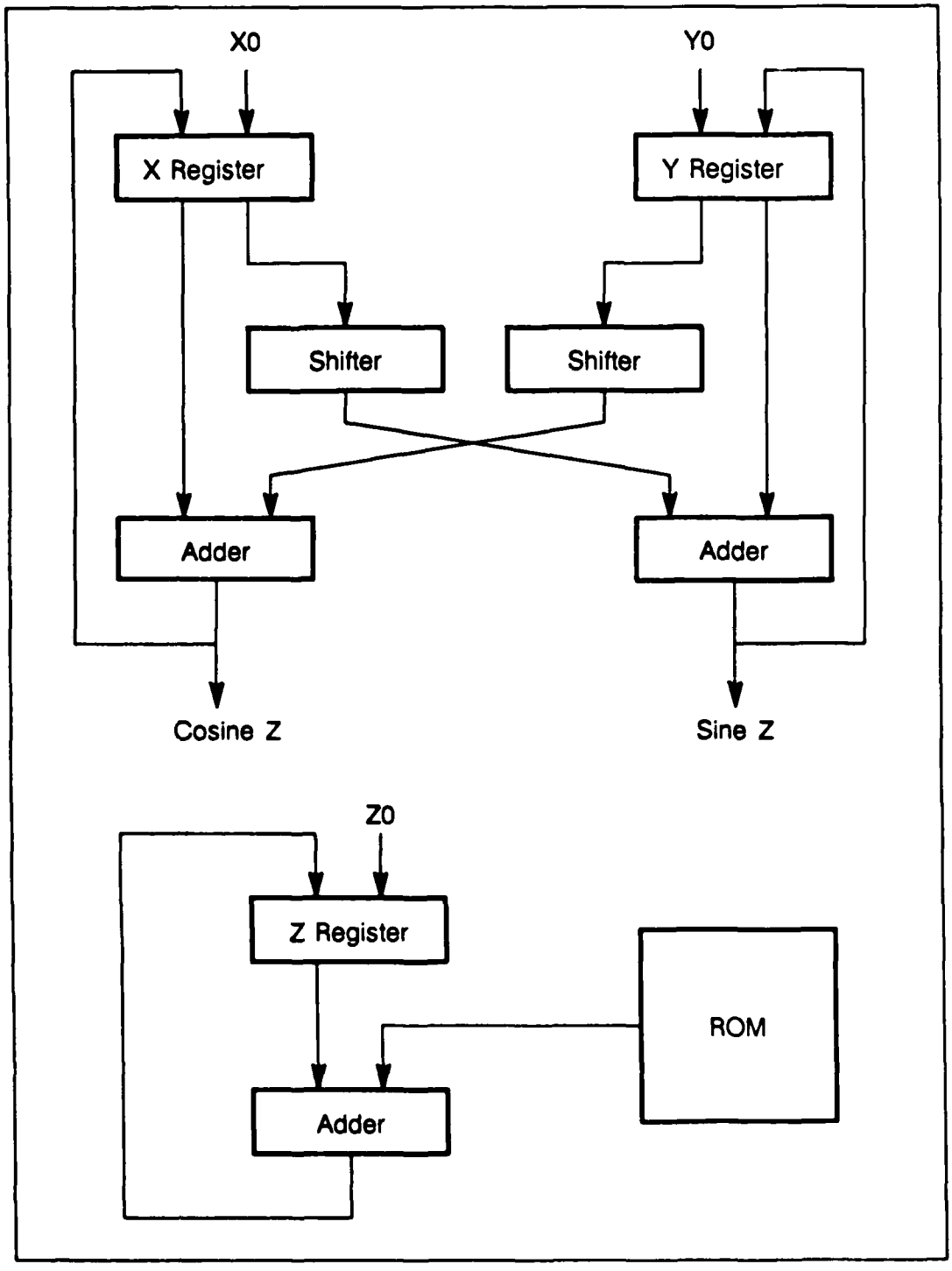


Figure 2.2. CORDIC Hardware for Sine and Cosine

each operation. A normalization is not required by fixed point adders and subtractors. Another reason fixed point hardware is used is that shifters are used in place of multipliers, because in binary, a multiply by 2^{-i} is the same as performing i right shifts in the fixed point format. Right shifts are not equivalent to multiplication in the floating point format.

CORDIC Division. The basic CORDIC equations can also be utilized to perform division as presented in (2:108). To use CORDIC for division,

$$Z_{i+1} = Z_i + (sign(Y_i)(2^{-i})) \tag{2.12}$$

$$X_{i+1} = X \tag{2.13}$$

$$Y_{i+1} = Y_i - sign(Y_i)\,(X)\,(2^{-i}) \tag{2.14}$$

where

$X_0 =$ divisor
$Y_0 =$ dividend
$Z_0 = 0$
$Z_{i+1} = Y_0/X_0$

Range reduction for division is easily accomplished since the input to the preprocessor is in the floating point format. Only the mantissa of the floating point number is used in the CORDIC processor. This greatly simplies the preprocessing as compared to that for sine and cosine. Since the VWE Processor only requires the reciprocal, then Y_0 is set equal to 1 and X_0 is set equal to the mantissa of the floating point number. The mantissa must be converted to the fixed point format.

The preprocessing of the exponent is to either add or subtract twice the unbiased exponent from the biased exponent of the argument, which then becomes the exponent of the quotient in the floating point format. If the exponent of the argument is positive, then the unbiased exponent is doubled and then subtracted from the exponent of the argument. If the

exponent of the argument is negative, then twice the magnitude of the unbiased exponent is added to the exponent of the argument.

The exponent and mantissa can be separated because the reciprocal of any number in scientific notation (and the floating point format is scientific notation in binary) is equal to the reciprocal of the mantissa times the inverse of the sign of the exponent. For example, the reciprocal of 1.2 E10 is equal to (1/1.2) E-10. But since the exponent is biased by 127, changing the sign of the exponent of the argument does not equal the reciprocal of the exponent. Instead, the unbiased exponent must be calculated by subtracting 127 from the exponent.

The hardware for the CORDIC division is the same as that for CORDIC sine and cosine, except that the extra hardware for the X's is not required (Figure 2.3). The ROM for division contains the values for 2^{-i}. The ROM could be replaced by a shifter.

Chebyshev Method

The theory of best approximation by polynomials was founded by P. L. Chebyshev (12:6). A polynomial, $P_n(x)$, can be found which is a best approximation of a function $f(x)$. An iterative process for constructing polynomials of the best approximation, $P_n(x)$, can be used to find approximations to the function $f(x)$ (12:7). The basis for these iterative equations is the Chebyshev polynominals, $T_n(x)$, which are defined as follows (5:48).

$$T_0(x) = 1 \tag{2.15}$$

$$T_1(x) = x \tag{2.16}$$

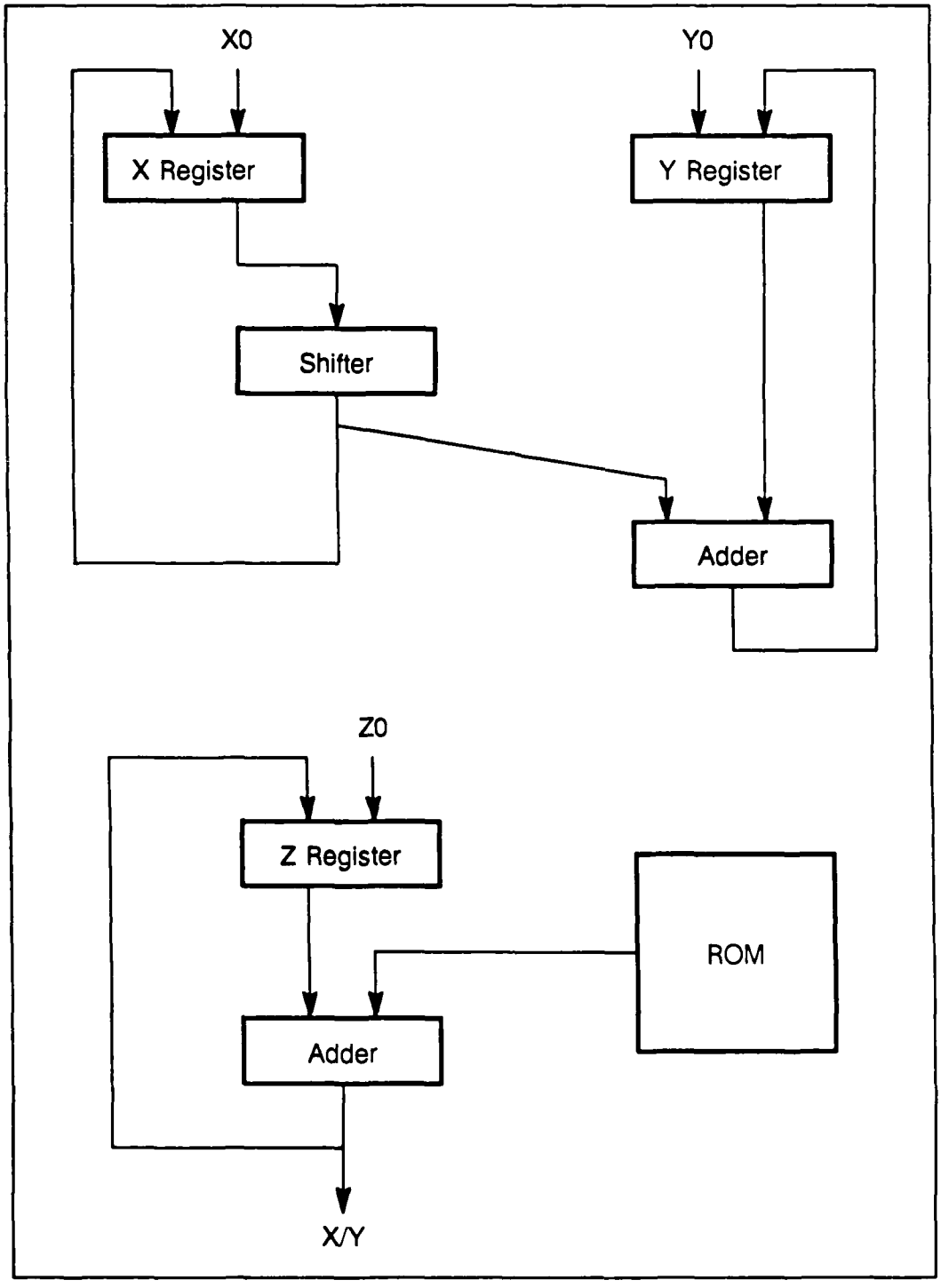


Figure 2.3. CORDIC Division Hardware

$$T_{n+1}(x) = (2 * x * T_n) - T_{n-1} \tag{2.17}$$

where $n \leq 2$.

The expansion of Chebyshev polynominals for the sine and cosine are introduced in (12:84-87). The following are the Chebyshev polynominal expansions for the sine and cosine functions:

$$\sin(\pi/2)x = x \sum_{k=0}^{\infty} a_k T_k(2x^2-1) \tag{2.18}$$

$$\cos(\pi/2)x = \sum_{k=0}^{\infty} a_k T_k(2x^2-1) \tag{2.19}$$

where $|x| \leq 0$.

Equations (2.18) and (2.19) both solve for $(\pi/2)x$. Substituting the minimum and maximum values for x in the equations reveals that the equations can be used to solve for $sin(z)$ or $cos(z)$, where z is in the range $[0, \pi/2]$. Even though the equations will provide answers for this range, the input range is still [0, 1]. This implies that both range normalization and scaling are necessary. Both of these will be discussed in the next section. The constants for the sine and cosine series are unique and are listed in Table 2.1.

Preprocessing and Range Reduction. The range of the arguments for the Chebyshev polynominals is [0, 1]. However, as mentioned before, the equations solve for $(\pi/2)x$. Therefore, both range reduction and scaling must be done to put α in the correct range. Scaling of the reduced argument is such that [0, 1] equates to $[0, \pi/2]$. Two methods are presented to perform this range reduction and scaling. The first method is a modified version of that for the range reduction for the CORDIC algorithm presented earlier. The second

Table 2.1. Constants for Chebyshev Sine and Cosine (12:87-88)

Constant	Sine	Cosine
a_0	1.276278962	0.472001216
a_1	-0.285261569	-0.499403258
a_2	0.009118016	0.027992080
a_3	-0.000136587	-0.000596695
a_4	0.000001185	0.000006704
a_5	-0.000000007	-0.000000047

method is from (9:123) and is another variation of the method presented for the CORDIC range reduction for the sine and cosine.

The first method is based on the range reduction to the range $[-\pi/2, \pi/2]$ which must be scaled to [0, 1]. Instead of folding the argument into the first or fourth quadrants, the argument is folded into the first quadrant. However, the required range is not the entire first quadrant which is $[0, \pi/2]$, but the range of the input to the Chebyshev equations is [0, 1]. The trigometric relationships

$$\cos t = \sin((\pi/2) - t) \tag{20}$$

and

$$\sin t = \cos((\pi/2) - t) \tag{21}$$

are used to process the input to the required range.

A portion of the scaling is performed by not multiplying the fractional part by π as before with the CORDIC range normalization. The remainder of the scaling is accomplished by multiplying the fractional part by 2. The range reduction and scaling have modified the Chebyshev equations from solving for $sin(\pi/2)x$ or $\cos(\pi/2)x$ to solving for $sin(x)$ or $cos(x)$.

The argument, α, is still multiplied by $1/\pi$, but now if the fraction part, F, is less than or equal to 0.5, y, the input to the Chebyshev equations, equals $2*F$. Otherwise, y is equal to $1-(2*F)$.

The determination of the sign of the final output of the Chebyshev Processor is more complicated than that for the CORDIC Processor. For the CORDIC range reduction, the quadrant the argument was in did not matter since the CORDIC equations compensated for this. However, since all values for the Chebyshev Processor are reduced to the first quadrant, then the quadrant the argument was in must be known.

For a positive argument, the sign for the sine of the argument will be positive if N, where N is the integer portion after multiplying the argument by $1/\pi$, is even because the sine function is positive in the first and second quadrants, quadrants which can be entered by an even multiple of π. Recall that F, the fractional portion after multiplying the argument by $1/\pi$, when multiplied by π, is the offset from either the beginning of the first quadrant or the third quadrant. The third and fourth quadrants correspond to odd multiples of π. Therefore, if N is odd, the argument was in either the third or fourth quadrants and the sine will be negative.

For a negative argument, the sign will be the negative of that above. The effect of the fractional part of a negative argument after multiplying by $1/\pi$ is to enter the quadrants from a clockwise rotation, instead of a counterclockwise rotation as for the positive argument.

The sign for the cosine of a positive argument is more complicated since the cosine is positive in the first and fourth quadrants but negative in the second and third quadrants. When N is even and $F \leq 0.5$, the argument was in the first quadrant. Recall that from Equation (2.8), F should also be multiplied by π. However, due to scaling considerations, this is not done. Before scaling, the 0.5 used in the inequality is the scaled equivalent of $\pi/2$. So,

an even N equates to an even multiple of π, which places the argument at the beginning of the first quadrant. All values in the first quadrant are between 0 and $\pi/2$. Therefore, when $F \leq 0.5$, the argument was orginally in the first quadrant. By the same reasoning, when N, the sign of the result is positive. Otherwise, the sign is negative since the argument was either in the second or third quadrants.

The effect of a negative argument is not the same for both the sine and cosine functions. A negative argument equates to a clockwise rotation through the quadrants instead of a counterclockwise rotation. For the sine function, if N is even, the rotation will be to the fourth and third quadrants where the sign is negative. Conversely, if N is odd, the rotation will be to the second and first quadrants where the sign is positive. However, the sign for a negative argument for the cosine function is the same as that for a positive argument since the cosine function is positive in the first and fourth quadrants and negative in the second and third quadrants. A clockwise or counterclockwise rotation will result in the same sign for the cosine function.

The steps for reducing the range to [0, 1] follows.

(1) Multiply α by $1/\pi$.

(2) Separate into integer, N, and fraction, F.

(3a) For Sine: If $F \leq 0.5$, then $y = 2 * F$ and compute the sine of y.

Otherwise, $y = 1 - (2 * F)$, and find cosine of y.

(3b) For Cosine: If $F \leq 0.5$, then $y = 2 * F$ and compute the cosine of y.

Otherwise, $y = 1 - (2 * F)$, and find sine of y.

(4a) For Sine: If N is even and positive, sign of the resultant is positive.

If N is odd and positive, the sign of the resultant is negative.

If N is even and negative, sign of the resultant is negative.

If N is odd and negative, the sign of the resultant is positive.

(4b) For Cosine: If $F \leq 0.5$ and N is odd, then sign is negative.

Otherwise, the sign is positive.

The method found in (9:123) uses the same basic equation but results in the use of different hardware.

(1) Compute $u = (2/\pi)x$.

(2) Compute $v = u - 4\lfloor (u+1)/4 \rfloor$.

(3) Set $z = v$ if $v \leq 1$.

Otherwise, $z = 2-v$.

(4) If $z \leq 0$, then sign equals -1.

Otherwise, sign equals 1.

(5) The magnitude of z is input to the Chebyshev Processor.

Chebyshev Processor Hardware. Equations (2.15) - (2.17) demonstrate the recurrance relation of the Chebyshev polynominals. This relation is used to design a processor using "pipeline networking" (9:121). A single stage of the pipeline for computing sine and cosine is shown in Figure 2.4. A single stage consists of 2 multipliers and 2 adders. The multiplier and adder on the left side of the figure are used to compute the sum of the product of the constants and the Chebyshev polynomials. The multiplier and adder on the right side of the figure are used to compute successive terms of the Chebyshev polynomials, $T_n(x)$.

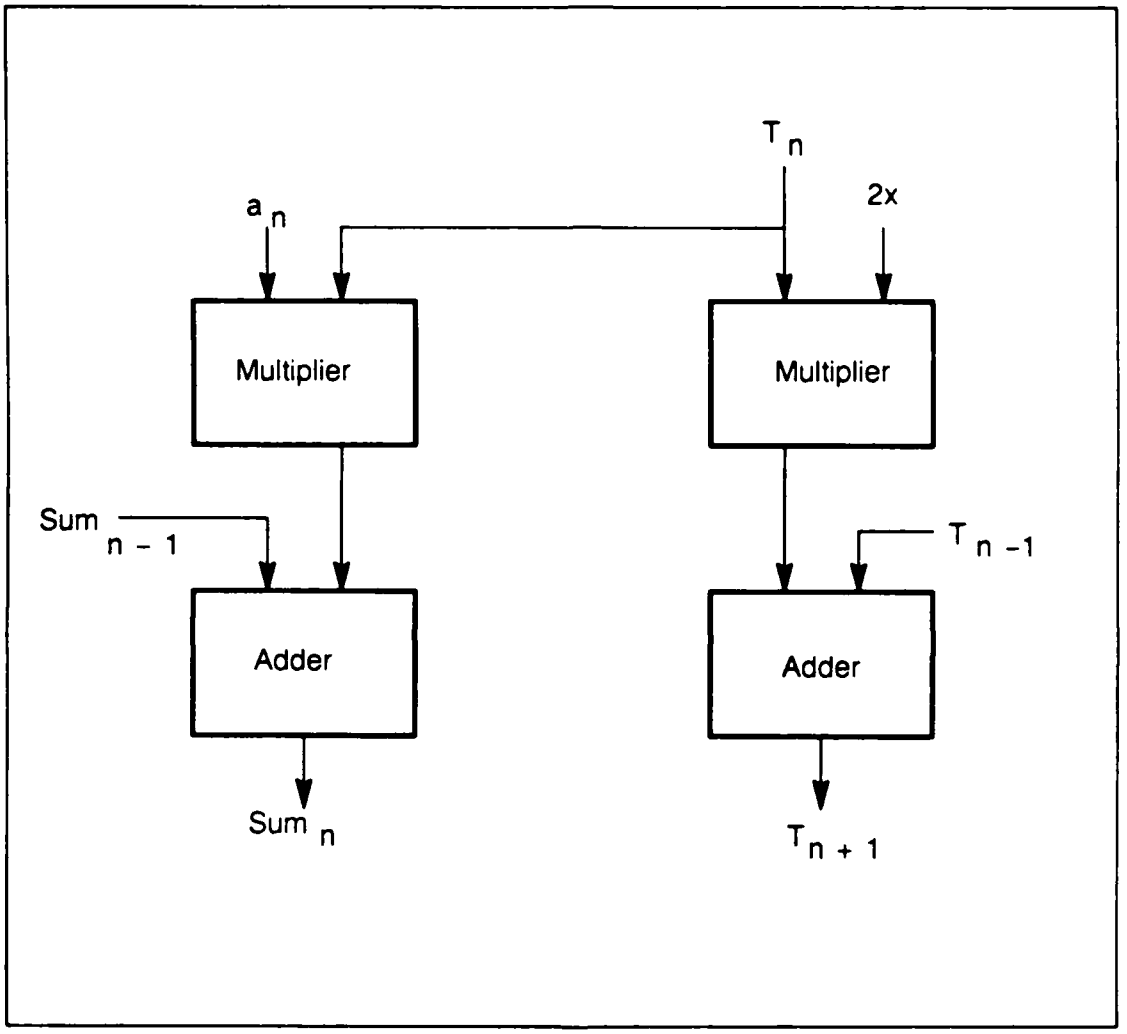


Figure 2.4. Single Stage of Chebyshev Processor (9:127)

The number of pipeline segments depends on the number of terms used in the summation. A total of 4 segments are required to meet IEEE Standard 754 for sine and cosine (9:124). The pipeline for the Chebyshev processor is shown in Figure 2.5. Note that since $T_0(x) = 1$, the output on the left side of the first multiplier/adder pair is the first 2 terms of the summation. As with the single stage shown in Figure 2.4, the left side of the figure computes the sum of the product of the constants and Chebyshev polynomials and the right side computes successive terms of the Chebyshev polynomials.

The preprocessing for range reduction as well as the postprocessing are also pipelined. The preprocessing for the sine and cosine are shown in Figure 2.6. The hardware shown in this figure is for the range reduction and scaling based on the method used for the CORDIC range reduction.

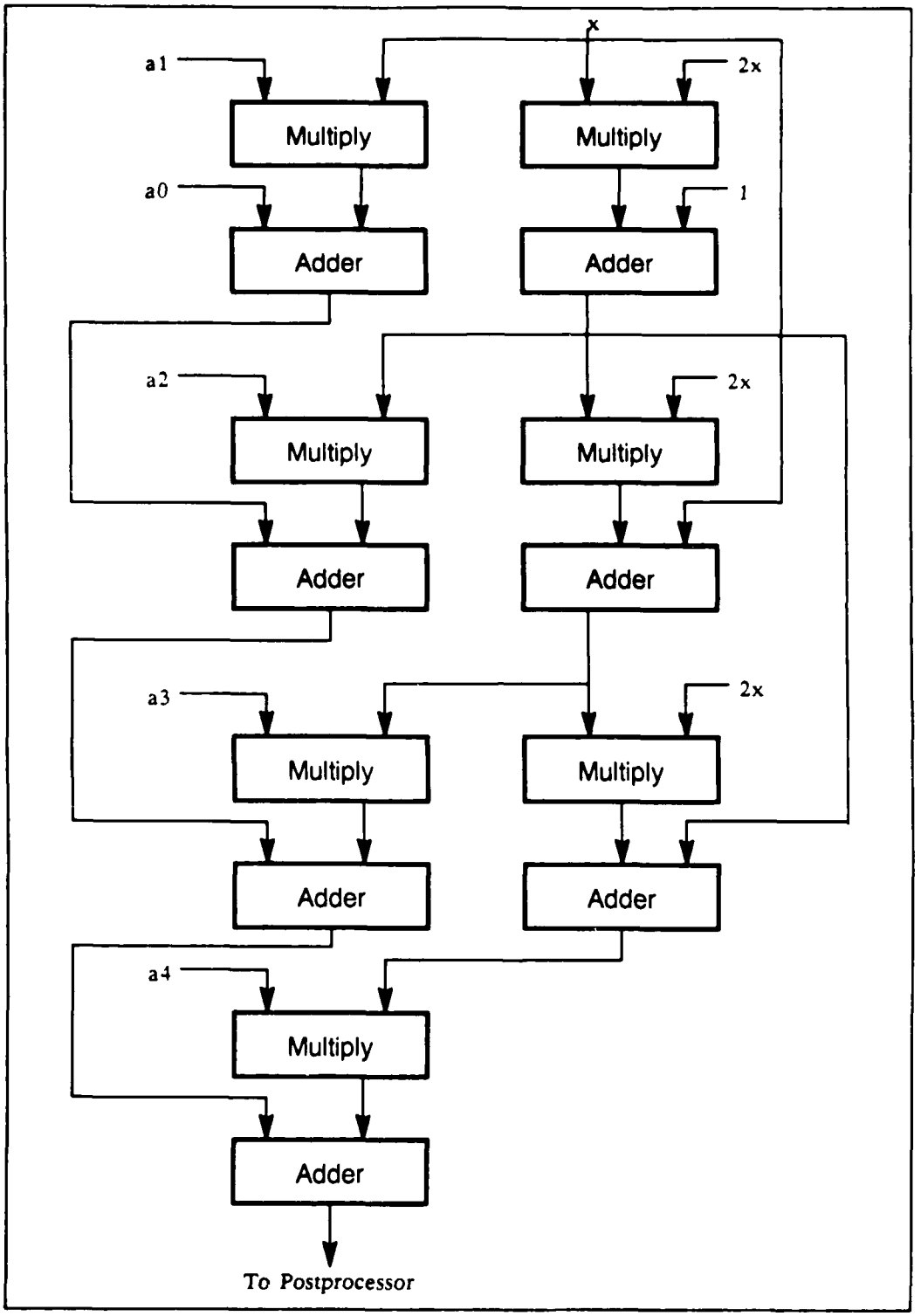


Figure 2.5. Chebyshev Processor Pipeline (9:127)

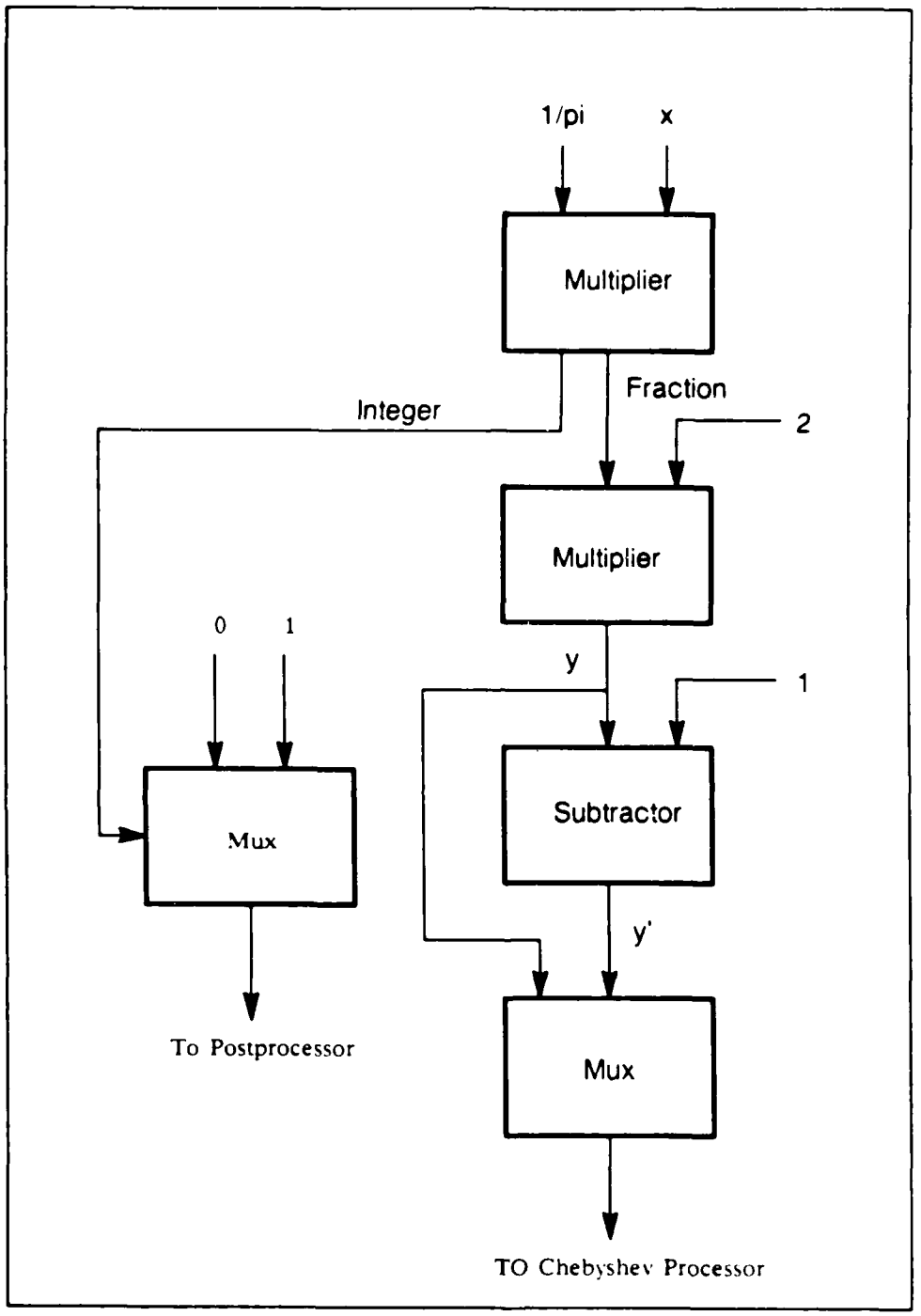


Figure 2.6. Preprocessing for Sine and Cosine

III. Expansion of Chebyshev Polynominals

The equations for the expansion of the Chebyshev polynominals for sine and cosine that were presented in Chapter II can be simplified by manipulation of the equation to polynomial form as a function of x instead of the Chebyshev polynominals, $T_n(x)$. A 25% reduction of hardware for each pipeline stage of the Chebyshev processor is possible. The equations as a function of T_n's require that the partial sum be computed in parallel with the successive T_n's. However, the same equation as a function of x does not require the computation of T_n, but only the powers of x. In particular, the equations as a function of x require either the even powers of x or the consecutive powers of x to be computed. Thus, an adder can be removed from each stage of the pipeline (See Figure 3.1).

An even greater reduction in hardware as well as a reduction in the time required to compute the functions are realized for Chebyshev polynominals which redefine the x of the Chebyshev polynominal. For example, consider the Chebyshev polynominal $T_n(2x^2-1)$. After any range reduction or scaling that may be necessary, x must be squared, doubled, and decremented by one before entering the Chebyshev processor. Thus, two extra multipliers and one extra adder/subtracter are added to the preprocessor. These extra steps are accounted for in the constants for the equations as a function of x.

Hardware reduction is also possible when the equations as a function of $T_n(x)$ require only the even or odd terms of the Chebyshev polynomials. Since the equations are iterative, consecutive $T_n(x)$'s must be computed regardless of whether only the even or odd terms are needed to compute a given transcendental elementary function. For single point precision, at least the first 5 terms of the Chebyshev polynomials are needed. If only the even values of $T_n(x)$ are needed, $T_2(x)$ through $T_8(x)$, then 8 pipeline stages will be required to compute the function. The same equation as a function of x, would only require 4 pipeline stages.

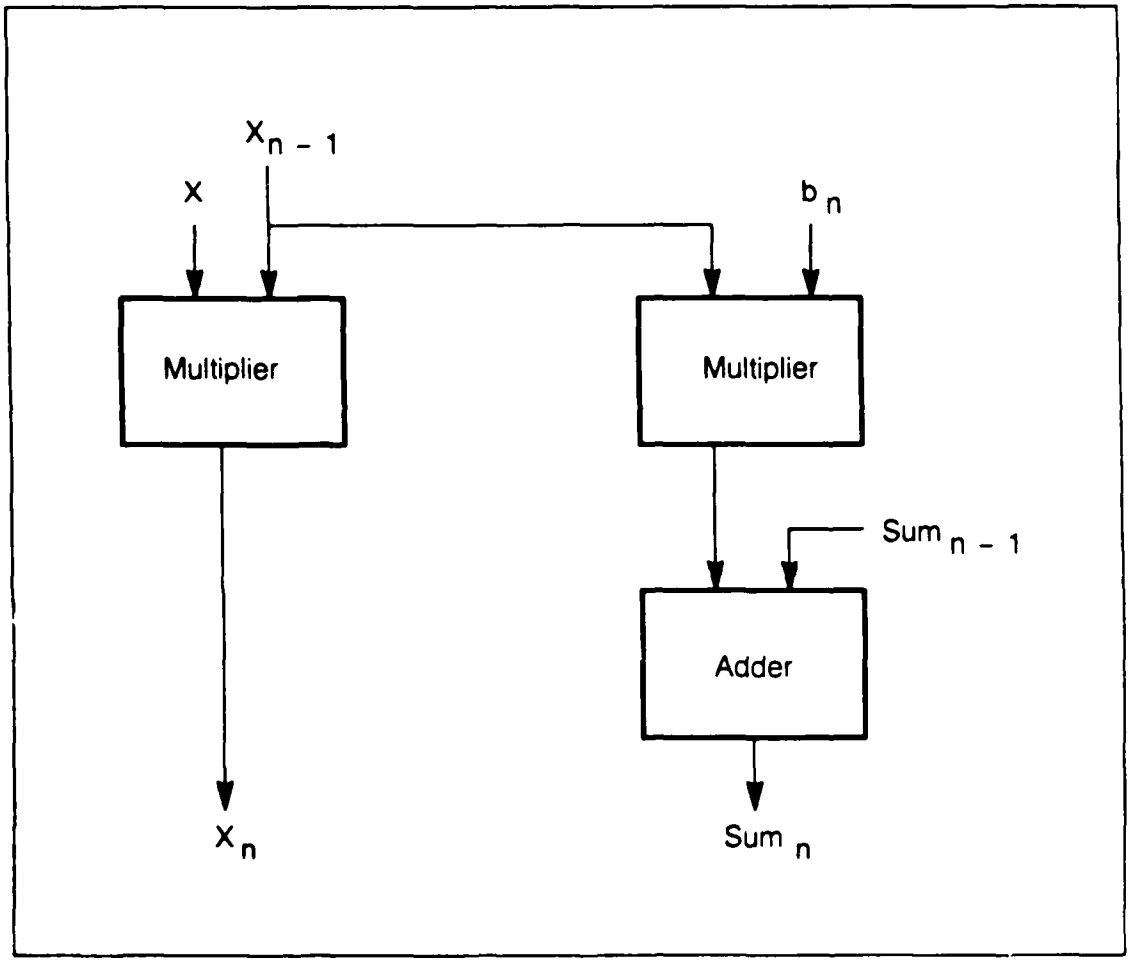


Figure 3.1. Single Stage of Chebyshev Processor as a Function of x

The constants for the equations as a function of $T_n(x)$, referred to as a_k in this thesis, do not depend on the number of iterations that will be performed. However, as a function of x, the constants, b_k, do depend on the number of iterations that will be performed, for each iteration adds another constant to each term of the series. That is, the number of terms needed in the Chebyshev polynominal must be considered before the b_k's are calculated. The number of terms is determined by the precision required for a particular application.

As stated in Chapter I, the Chebyshev polynomials can be used to compute functions other than sine and cosine. In this chapter, Chebyshev polynomials to compute tangent, arctangent, exponential, and natural logarithm will be discussed. The equations as functions of the Chebyshev polynomials as well as x and the constants, a_k's and b_k's, as well as the preprocessing and postprocessing hardware will be presented.

Sine and Cosine

The equation for the cosine function using Chebyshev polynominals can be expanded and simplified as a function of x with different coefficients as follows. The equation for the cosine for single precision is

$$\cos(pi/2)x = \sum_{k=0}^{4} a_k T_k(2x^2-1) \tag{3.1}$$

Substituting T'_k for $T_k(2x^2-1)$ and expanding Equation (3.1) results in

$$\cos(pi/2)x = a_0T'_0 + a_1T'_1 + a_2T'_2 + a_3T'_3 + a_4T'_4 \tag{3.2}$$

where

$T'_0 = 1$
$T'_1 = 2x^2-1$

$$T'_2 = 8x^4-8x^2+1$$
$$T'_3 = 32x^6-48x^4+18x^2-1$$
$$T'_4 = 128x^8-256x^6+160x^4-32x^2+1$$

Substituting the values for T'_k in Equation (3.2) results in

$$\cos(pi/2)x = a_0+a_1(2x^2-1)+a_2(8x^4-8x^2+1)+a_3(32x^6-48x^4+18x^2-1) + a_4(128x^8-256x^6+160x^4-32x^2+1) \quad (3.3)$$

Expanding all the terms in Equation (3.3) results in

$$\cos(pi/2)x = a_0+2a_1x^2-a_1+8a_2x^4-8a_2x^2+a_2+32a_3x^6-48a_3x^4 + 18a_3x^2-a_3+128a_4x^8-256a_4x^6+160a_4x^4-32a_4x^2+a_4 \quad (3.4)$$

And finally, grouping like terms of x results in

$$\cos(pi/2)x = (a_0-a_1+a_2-a_3+a_4)+(2a_1-8a_2+18a_3-32a_4)x^2 + (8a_2-48a_3+160a_4)x^4+(32a_3-256a_4)x^6+128a_4x^8 \quad (3.5)$$

By substituting the values from Table 2.1 for the a_k s in the above equation, the equation is now a function of x with new constants, b_k such that

$$\cos(pi/2)x = \sum_{k=0}^{4} b_{2k}\, x^{2k} \quad (3.6)$$

where

$b_0 = 0.9999999953$
$b_2 = -1.233698208$
$b_4 = 0.253650711$
$b_6 = -0.020810573$
$b_8 = 0.000858163$

The development of the sine as a function of x is similar and follows:

$$\sin(\pi/2)x = x \sum_{k=0}^{4} a_k T_k(2x^2-1) \tag{3.7}$$

Substituting T'_k for $T_k(2x^2-1)$ and expanding Equation (3.7) results in

$$\sin(pi/2)x = a_0xT'_0+a_1xT'_1+a_2xT'_2+a_3xT'_3+a_4xT'_4 \tag{3.8}$$

Substituting in the values for T'_k, expanding the equation, and grouping together terms of x results in the following equation:

$$\sin(pi/2)x = (a_0-a_1+a_2-a_3+a_4)x+(2a_1-8a_2+18a_3-32a_4)x^3 + (8a_2-48a_3+160a_4)x^5+(32a_3-256a_4)x^7+128a_4x^9 \tag{3.9}$$

By substituting the values from Table 2.1 for the a_k's in the above equation, the equation is now a function of x with new constants, b_k, such that

$$\sin(pi/2)x = x\sum_{k=0}^{4} b_{2k}\, x^{2k} \tag{3.10}$$

where

$b_0 = 1.570796290$
$b_2 = -0.645963360$
$b_4 = 0.079688480$
$b_6 = -0.004627223$
$b_8 = 0.000150820$

The constants, b_k's, for the sine and cosine expansions of x are listed in Table 3.1. These constants are for the expansion of the first 5 terms as required for single precision.

Table 3.1. Constants for Chebyshev Sine and Cosine as a Function of x

Constant	Sine	Cosine
b_0	1.570796290	0.999999953
b_2	-0.645963360	-1.233698208
b_4	0.079688480	0.253650711
b_6	-0.004627223	-0.020810573
b_8	0.000150820	0.000858163

This expansion allows for an architectural pipeline implementation. The pipeline for the expansion of the Chebyshev polynominals for the sine and cosine as a function of x is shown in Figure 3.2. The significant difference between this figure and Figure 2.4 is that one less adder is required per stage. This is a hardware savings of approximately 25%.

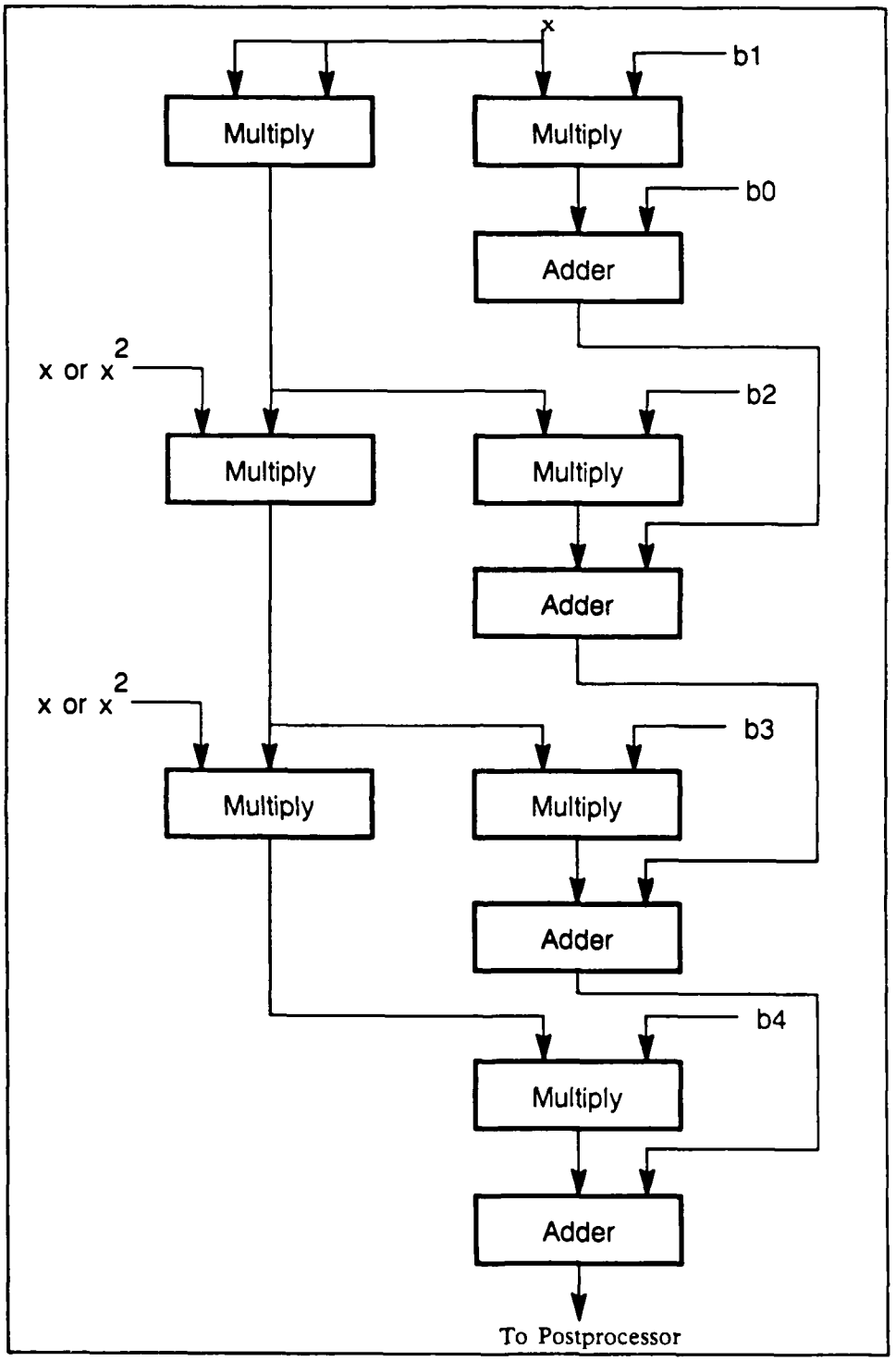


Figure 3.2. Chebyshev Processor as a Function of x

Tangent

The formula for the Chebyshev approximation of tan (x) is as follows (11:87):

$$\tan(pi/4)x = \sum_{k=0}^{\infty} a_{2k+1}\, T_{2k+1}(x) \tag{3.11}$$

where $(|x| \leq 1)$.

Computer simulation of this equation revealed that the first 6 terms of the summation are required for single precision. The a_{2k+1} are listed in Table 3.2.

Table 3.2. Constants for Chebyshev Tangent Function (11:87)

k	a_{2k+1}
0	0.93845067562
1	0.05717001507
2	0.00406513598
3	0.00029161838
4	0.00002093559
5	0.00000150310

Following the same steps as for the sine and cosine, the Chebyshev expansion as a function of x is as follows.

$$\tan(pi/4)x = \sum_{k=0}^{5} b_{2k+1}\, x^{2k+1} \tag{3.12}$$

However, to simplify the pipeline so that only the even powers of x are required, the equation can be changed to the following:

$$\tan(pi/4)x = x \sum_{k=0}^{5} b_{2k} x^{2k} \tag{3.13}$$

where

$b_0 = 0.78539686786$
$b_2 = 0.16152638116$
$b_4 = 0.03957327280$
$b_6 = 0.01083740608$
$b_8 = 0.00112678144$
$b_{10} = 0.00153917440$

Preprocessing for Tangent. Given that Equation (3.11) computes tan(x), and the range of x is [0, 1], only the tangent of [0, $\pi/4$] is available from the equation. Unlike sine and cosine, the magnitude of all values of the tangent can be found between 0 and $\pi/2$. Two methods exist to obtain the tangent of inputs in the range [$\pi/4$, $\pi/2$):

Find the cotangent since

$$\tan(x) = cot((\pi/2)-x) \tag{3.14}$$

or subtract x from $\pi/2$ and invert the output of the processor since

$$\tan((\pi/2)-x) = 1/\tan(x) \tag{3.15}$$

Both methods would require a division. The Chebyshev polynomials for the cotangent solves for $xcot(x)$, so a division by x is required to find $cot(x)$. The range reduction is the same for both methods, the only difference is the Chebyshev equation used to solve for tan (x). Finding the cotangent is more desirable as a function of hardware, since the tangent method requires a division in the postprocessing phase. For the cotangent equation, the reciprocal of x can be computed in parallel with the Chebyshev processor, so that only a multiply is required in the postprocessor. A multiplier is already required by the sine function in the postprocessor. The Chebyshev polynomial expansion for the cotangent is as follows (11:87):

$$x \cot (pi/4)x = \sum_{k=0}^{\infty} a_{2k} T_{2k}(x) \tag{3.16}$$

Computer simulations for Equation (3.16) revealed that the first 6 terms are necessary for single precision. The constants, a_{2k}'s for cotangent (x) are listed in Table 3.3.

The same equation as a function of x is

$$x \cot (pi/4)x = \sum_{k=0}^{5} a_{2k} x^{2k} \tag{3.17}$$

where

$b_0 = 1.27321935363$
$b_2 = -0.26143595267$
$b_4 = -0.01173517139$
$b_6 = 0.00001332237$
$b_8 = -0.00003841664$
$b_{10} = -0.00000294400$

The steps for reducing the range to [0, 1] take into account the symmetry of the tangent function. As with the range reduction for sine and cosine, after the multiplication by $1/\pi$, the

Table 3.3. Constants for Chebyshev Cotangent Function (11:87)

k	a_{2k}
0	0.93845067562
1	0.05717001507
2	0.00406513598
3	0.00029161838
4	0.00002093559
5	0.00000150310

product is separated into an integer, N, and fraction, F. Due to the scaling, F will be multiplied by 4 instead of 2. However, further reduction of F is necessary if F is greater that 0.25 since $|x| \leq 1$. The range that can be solved by the Chebyshev equation for the tangent is $[0, \pi/4]$. Since $\tan(x) = \tan(\pi - x)$, two possible ranges for the F before the multiplication by 4 are considered as inputs to the Chebyshev tangent equation: $x \leq 0.25$ or $x \geq 0.75$. For $x \leq 0.25$, no further reduction is necessary. But if $x \geq 0.75$, then x must be subtracted from 1.

For x in the range (0.25, 0.75), the Chebyshev equation for the cotangent will be used as $\tan(x) = cot((\pi/2)-x)$ and $cot(x) = cot(\pi - x)$. If x is in the range (0.25, 0.5], x is subtracted from 0.5. If x is in the range (0.5,0.75], then 0.5 is subtracted from x.

The following steps are used to reduce the input to the required range. The resulting hardware is shown in Figure 3.3.

(1) Multiply α by $1/\pi$.

(2) Separate into integer, N, and fraction, F.

(3) If $F \leq 0.25$, then $x = 4 * F$ and compute the tangent of x.

If $0.25 < F \leq 0.5$, $x = 4(0.5 - F)$, and find cotangent of x.

If $0.5 < F \leq 0.75$, $x = 4(F - 0.5)$, and find cotangent of x.

If $F \geq 0.75$, then $x = 4(1 - F$ and compute the tangent of x.

(4) If N is even and $\alpha \geq 0$, sign = 1.

If N is even and $\alpha < 0$, sign = -1.

If N is odd and $\alpha \geq 0$, sign = -1.

If N is odd and $\alpha < 0$, sign = 1.

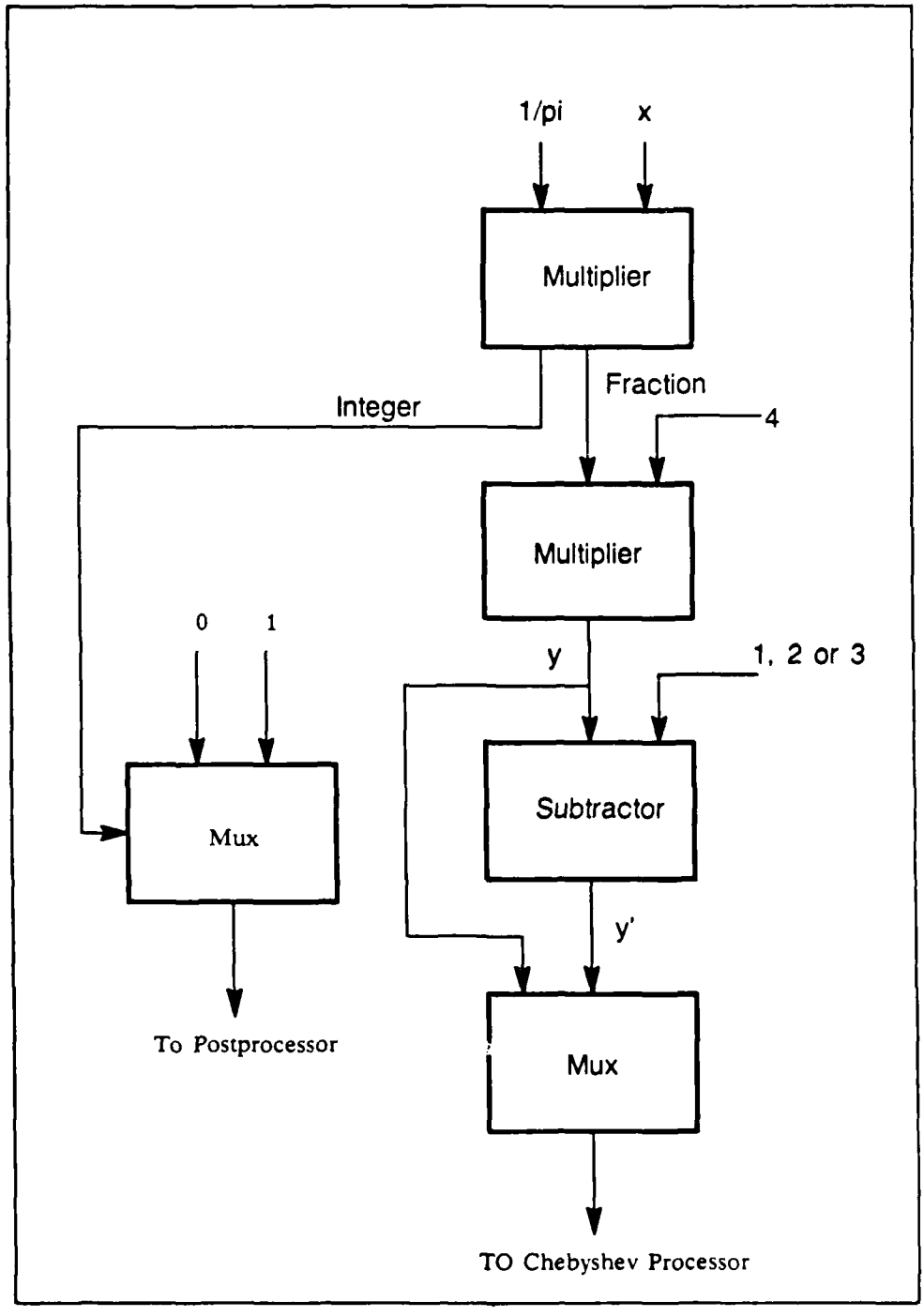


Figure 3.3. Preprocessing for Tangent

Postprocessing for Tangent.. The postprocessing for the tangent function consists of correcting the sign of the output of the Chebyshev processor. If the Chebyshev cotangent equation is used, the output must be multiplied by $1/x$.

Arctangent

The expansion of the Chebyshev polynominal for the arctangent (11:111) is

$$\tan^{-1}(x) = x \sum_{k=0}^{\infty} a_k T_k(2x^2-1) \tag{3.18}$$

Computer simulation of Equation (3.18) revealed that the first 5 terms of the equation are required for single precision. The constants for Equation (3.18) are listed in Table 3.4.

Table 3.4. Constants for Chebyshev Arctangent Function (11:111).

k	a_k
0	1.570796290
1	-0.645963360
2	0.079688480
3	-0.004627223
4	0.000150820

The same equation as a function of x is as follows:

$$\tan^{-1}(x) = x \sum_{k=0}^{4} b_{2k} x^{2k} \tag{3.19}$$

where

$b_0 = 0.99999990304$
$b_2 = -0.33332184597$
$b_4 = 0.19961556588$
$b_6 = -0.13751623066$
$b_8 = 0.07726269923$

Range Reduction for Arctangent. The range reduction for the arctangent presents a problem: a division is necessay if the magnitude of the original argument is greater than 1. If the magnitude of the argument is greater than 1, then the trigonometric identity

$$\tan^{-1}(x) = \pi/2 - \tan^{-1}(1/x) \tag{3.20}$$

must be used. Since

$$\tan^{-1}(x) = sign(x) \tan^{-1}(x) \tag{3.21}$$

no problem is encountered in finding the arctangent of positive or negative arguments.

The only two steps for range reduction follow. The range reduction hardware is shown in Figure 3.4.

(1) If the argument is negative, Sign equals -1.

(2) If $|x|$ is greater than 1, compute $1/x$. Use $1/x$ as the input to the Chebyshev processor.

Postprocessor for Arctangent. The postprocessing for the arctangent function has two possible steps. One is to correct the sign of the output if the argument was negative. The other step is to subtract the result from $\pi/2$ if the magnitude of the argument was greater than 1.

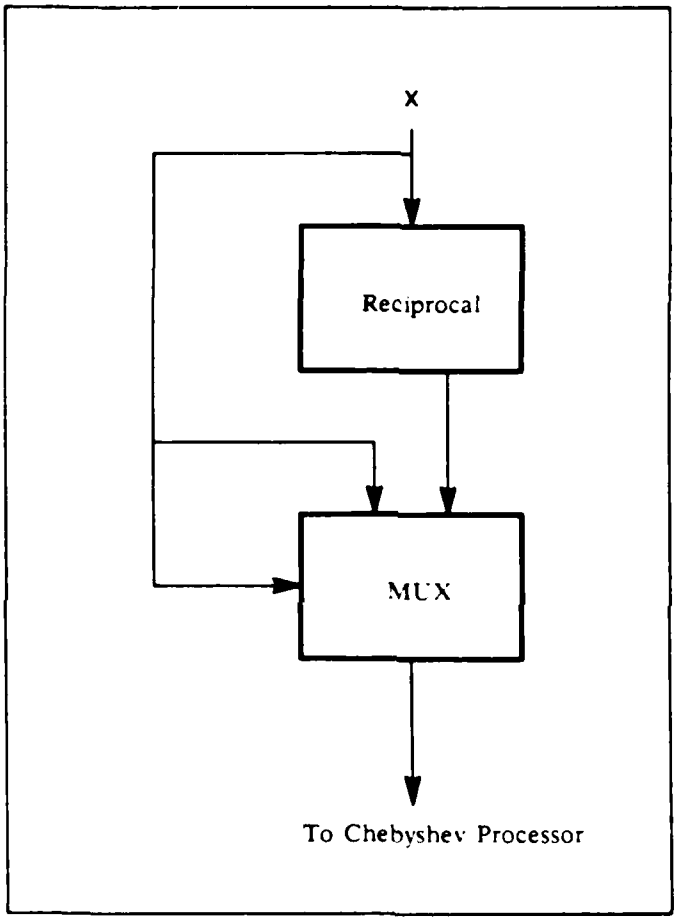


Figure 3.4. Preprocessing for Arctangent

Exponential

Two possible methods for the exponential are available. For the first method, two expansions for the Chebyshev polynomial for the exponential are required. One is needed for a positive exponential and the other is for a negative exponential. The two equations for the expansion of the exponential series follow (11:37-38).

$$e^{x} = \sum_{k=0}^{\infty} a_k T_k(x) \tag{3.22}$$

$$e^{-x} = \sum_{k=0}^{\infty} a_k T_k(2x-1) \tag{3.23}$$

Computer simulations for Equations (3.22) and (3.23) revealed that the first 6 terms are required for single precision. The constants, a_k's for exponential are listed in Table 3.5.

Table 3.5. Constants for Chebyshev Exponential Function (11:37-38).

a_k	e^{x}	e^{-x}
0	1.06348337074	0.645035270
1	0.25789430539	-0.312841606
2	0.03190614918	0.038704116
3	0.00264511197	-0.003208683
4	0.00016480555	0.000199919
5	0.00000822317	-0.000009975

As a function of x, the corresponding equations for the exponential are

$$e^x = \sum_{k=0}^{5} b_k x^k \tag{3.24}$$

where

$b_0 = 1.00000003902$
$b_1 = 0.99999621191$
$b_2 = 0.50005986425$
$b_3 = 0.16631361911$
$b_4 = 0.04264973595$
$b_5 = 0.00695192477$

and

$$e^{-x} = \sum_{k=0}^{5} b_k x^k \tag{3.25}$$

where

$b_0 = 0.99999994524$
$b_1 = -0.69314320166$
$b_2 = 0.24017948944$
$b_3 = -0.05529960672$
$b_4 = 0.00921087488$
$b_5 = -0.00947553245$

The other equation for computing the exponential makes use of the identity

$$e^x = 2^{x(ln_2 e)} = 2^r * 2^s \tag{3.25}$$

where

r = *Integer portion of* $x(ln_2 e)$
s = *Fraction of* $x(ln_2 e)$

The equation as a function of the Chebyshev polynomials is as follows (11:37):

$$2^x = 2^{1/2}[I_0((1/2)ln(2)+2\sum_{k=1}^{\infty} I_k((1/2)ln(2))T_k(2x-1)] \tag{3.26}$$

where

$I_0((1/2)ln(2)) = 1.03025449181$
$I_1((1/2)ln(2)) = 0.17590160392$
$I_3((1/2)ln(2)) = 0.01516500518$
$I_4((1/2)ln(2)) = 0.00087378181$
$I_5((1/2)ln(2)) = 0.00000130864$
$I_6((1/2)ln(2)) = 0.00000003777$
$I_7((1/2)ln(2)) = 0.00000000002$

However, Equation (3.26) is not accurate enough for single precision. Only 2 to 3 decimal places of accurary can be obtained with this equation.

Range Reduction for Exponential. For Equations (3.22) and (3.23) the range reduction makes use of the identity

$$e^x = e^{N+F} = e^N e^F \tag{3.27}$$

where N is the integer part and F is the fraction part. The values for e^N are stored in a ROM table. The F is fed into the Chebyshev processor and the result from the processor is multiplied by the value from the ROM table in the postprocessor.

For single precision and positive values of x, values for e^N, for N in the range [0, 88] must be stored in the ROM table. The range is determined by the largest value of e^N which can be represented in single precision. The largest number that can be represented is equal to 2^{128} which equals 3.4 E38. The natural logarithm of 3.4 E38 equals approximately 88.7. So, e^{88} is the largest integer value that can be represented in single precision.

For a negative value of x, the ROM table must contain values in the range [−88, 0]. This time the range is determined by the smallest value of e^N which can be represented in single precision. The smallest number is equal to 2^{-127} which approximately equals 5.9 E-39. The natural logarithm of 5.9 E-39 equals approximately -88.03. So, e^{-88} is the smallest integer value that can be represented in single precision.

The following steps are used to reduce the input for the exponential function using the equations for e^x and e^{-x}. The range reduction hardware is shown in Figure 3.5.

(1) Separate into integer N and fraction F.

(2) Input F into Chebyshev processor.

(3) Lookup value in ROM table corresponding to N to obtain e^N. Send to postprocessor.

Postprocessor for Exponential. The only step in the postprocessor is to multiply the output from the Chebyshev processor by the value from the ROM table.

Natural Logarithm

The expansion for the natural logarithm is rather complicated as a function of T_n as can be seen in Equation (3.27). However, as a function of x, the equation is rather simple and eliminates much of the preprocessing that would be required to obtain the $T_n(x)$s. The equation for the natural logarithm is (11:58)

$$\ln(1+x) = \sum_{k=0}^{\infty} a_k T_k(1+(4+2(2^{1/2}))x) \tag{3.28}$$

where $(2^{1/2}/2) - 1 \le x \le 0$.

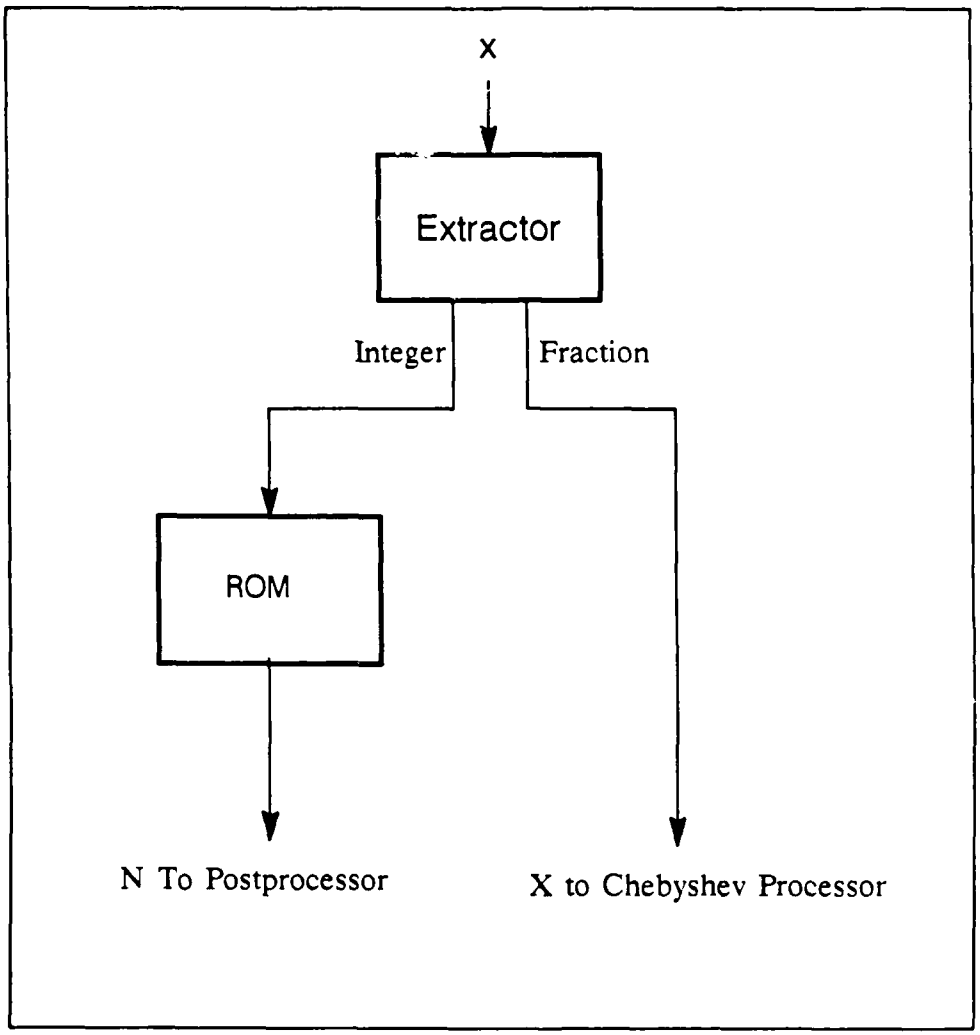


Figure 3.5. Preprocessing for Exponential

Computer simulation of Equation (3.28) reveals that the first 6 terms are required for single precision. The constants, a_k's for ln (1+x) are listed in Table 3.6.

Table 3.6. Constants for Chebyshev Natural Logarithm Function (11:58)

k	a_k
0	-0.16578909074
1	0.17285446745
2	-0.00746966673
3	0.00043038842
4	-0.00002789796
5	0.00000192891

The same equation as a function of x is as follows.

$$\ln (1+x) = \sum_{k=0}^{5} b_k x^k \tag{3.29}$$

where

$b_0 = -0.00000000958$
$b_1 = 0.99999686448$
$b_2 = -0.50016647931$
$b_3 = 0.33005325226$
$b_4 = -0.28021885649$
$b_5 = 0.06217418014$

Range Reduction for Natural Logarithm. The range reduction for this function is complicated but greatly simplified by the binary floating point format. The following logarithmic identies are used for the range reduction.

$$\ln(mn) = \ln(m) + ln(n) \tag{3.30}$$

$$\ln(p^q) = q * \ln(p) \tag{3.31}$$

$$\ln(r/s) = ln(r) - ln(s) \tag{3.32}$$

The range for the input is $-0.292893219 \leq x \leq 0$. Since the Chebyshev polynominal computes ln (1+x), to find ln (x), x+1 must be in the range $.707106781 \leq x+1 \leq 1$. For reasons to be discussed in the following paragraphs, the range is further reduced to $0.75 \leq x+1 \leq 1$.

The first step makes use of Equation (3.30), where m is the exponent and n is the mantissa of the argument, α. The first step is to extract the exponent and the mantissa of α. Then m becomes the exponent of a new floating point number, A, where the 23 bits of the mantissa of A are all zeros and n becomes the mantissa of a new number, B, with an exponent equal to 0.

The floating point format is such that a number is normalized, which means that there is an implied 1 point before the mantissa, so B is greater than 1. But the range of x must be less that 1. To make this appear as if the leading 1 is actually the first digit after the decimal, the exponent of B is decremented by 1. To maintain the equality, the exponent of A must be incremented by 1.

In reality, decrementing the exponent of B does not change the mantissa. But for the purposes of the range reduction algorithm, the first bit after the binary point can now be a 1, which means that the smallest the number can be (in decimal) is 0.5. Based on the value of the next two bits, which are the first two bits of the 23 bits allocated for the mantissa, one of three constants will be selected to form a product of the constant and B which will be in the required range.

Equation (3.31) is used to find the natural logarithm of A since A is equal to 1.0×2^m. In terms of Equation (3.31), $\ln(A) = m\ln(2)$ or $A * \ln(2)$.

Finally, Equation (3.32) is used to insure the input into the Chebyshev processor is in the proper range. For $0.5 \leq x < 0.625$, if x is multiplied by 1.5, then x will be in the proper range. And for $0.625 \leq x < 0.75$, x times 1.2 will place x in the proper range. To compensate for the multiplication by either 1.2 or 1.5, a division by either 1.2 or 1.5 is required to maintain the equality. However, the division would be part of the postprocessing, after the natural logarithm of $(1+x)$ has been calculated. The multiplication by either 1.2 or 1.5 in the preprocessor can now be seen as a multiply by the natural logarithm of 1.2 or 1.5. To maintain the equality, a division by the natural logarithm of either of 1.2 or 1.5 is required. By Equation (3.32), this division can be a subtraction.

The following steps are required to solve for the $ln(x)$ and the hardware is shown in Figure 3.6.

(1) Separate input into exponent, m, and mantissa, n. Put each into new floating point numbers such that A has an exponent equal to m and a mantissa equal to 1.0 and B has an exponent equal to 0 and the mantissa equal to n.

(2) Multiply A by 2 and B by 1/2.

(3) If the most significant bit (MSB) of the mantissa of B, not including the implied 1 point, is a 1, then $x = (B * 1)-1$.

If the MSB of B equals zero, then if (MSB - 1) equals 1, $x = (B * 1.2)-1$.

If the MSB and (MSB - 1) equal 0, then $x = (B * 1.5)-1$.

(4) Input x into the Chebyshev processor.

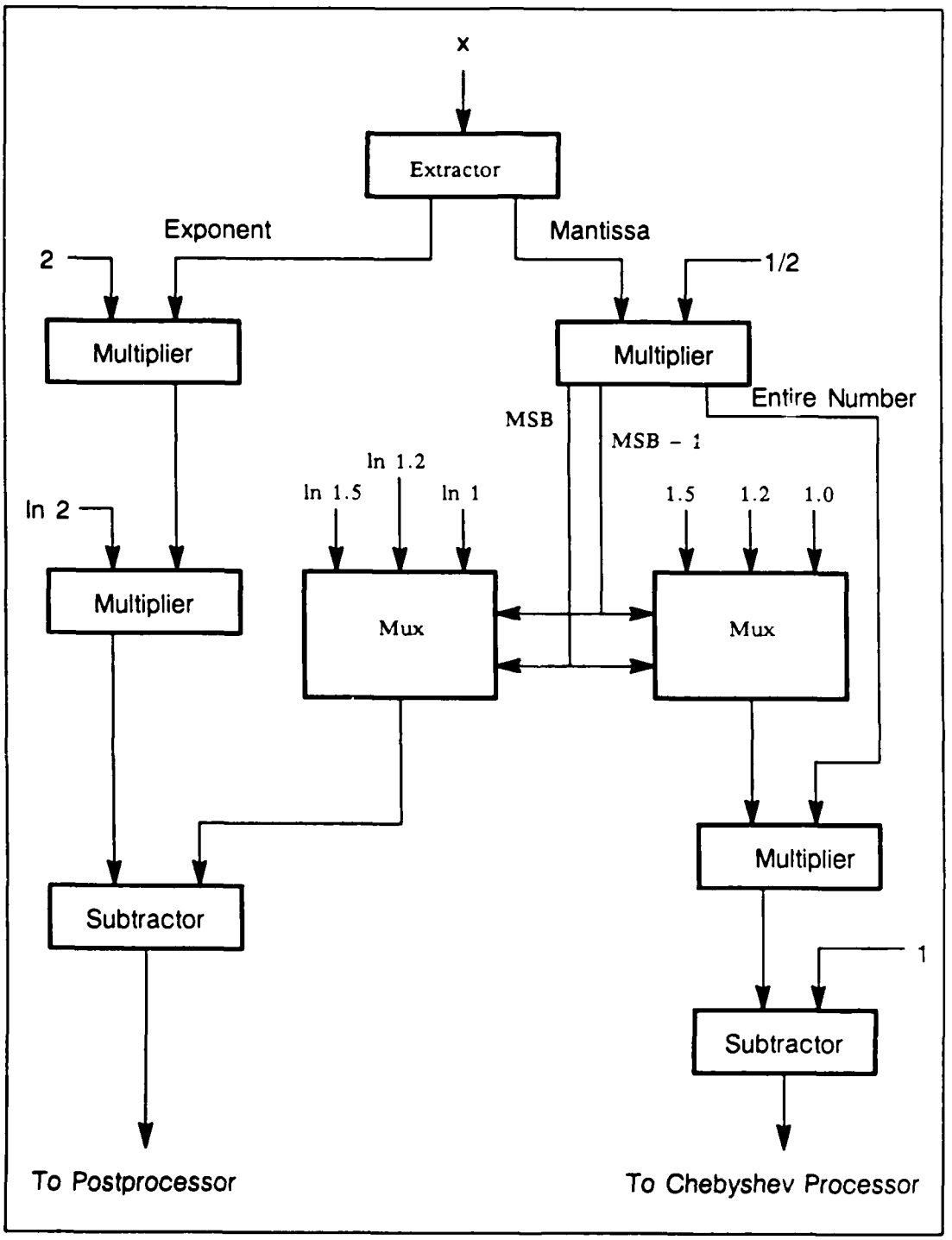


Figure 3.6. Preprocessing for Natural Logarithm

(5) Compute $C = A * \ln(2)$.

(6) If constant in (3) equals 1, subtract ln(1) from C.

If constant in (3) equals 1.2, subtract ln(1.2) from C.

If constant in (3) equals 1.5, subtract ln(1.5) from C.

(7) Send output from (6) to postprocessor.

Postprocessor for Natural Logarithm. The only step in the postprocessor is to add the value from the preprocessor to the output of the Chebyshev processor.

IV. Analysis of Chebyshev and VWE Processors

The results of the previous two chapters are analyzed in this chapter. First, the times for the fastest hardware, including a multiplier and adder, will be discussed. Second, the number of pipeline stages for the Chebyshev processor will be considered. Also, a unified pipeline for the preprocessor and postprocessor will be presented. Finally, analysis of the functions for the times for the VWE processor will be presented.

State of the Art Hardware

Before any analysis of the processors can be done, computation times for the basic hardware units must be presented. The faster these units can compute, the greater the speedup of the processors. The fastest computation times found for the adder and multiplier are those developed at the Air Force Institute of Technology and were provided by (6). The times are presented in Table 4.1.

Table 4.1. State of the art hardware (6)

Unit	Time(ns)
Floating Point Multiply	80
Floating Point Adder	50
Fixed Point Adder	35
Shifter	5

Chebyshev Processor Analysis

As discussed in the previous chapter, when the expansion of the Chebyshev polynominals is expressed as a function of x, the constants, b_k's, change, depending on the required number of iterations.

To determine the b_k's for single precision, all of the required functions were simulated in Pascal. The computer programs are listed in the appendix. For single precision, 2^{-24} or 7 to 8 decimal places of accuracy are needed. The number of terms required for each of the elementary functions are listed in Table 4.2.

Table 4.2. Number of iterations for Chebyshev processor.

Function	Number Iterations
sine	5
cosine	5
tangent	6
cotangent	6
ln	5
arctan	5
exp	5

Since the design of the Chebyshev processor requires a pipeline for the evaluation of the Chebyshev polynominals, 6 iterations will be required. This will require 5 pipeline stages. As the first term of the summation is a multiply of the constant, b_0 times 1, the multiplication is not necessary. The first step of the first stage of the pipeline is a multiply by the

second constant, b_1 with x. The first constant, b_0 is then added to the product in the last step of the first stage of the pipeline.

Each stage in the pipeline will take 160 ns to compute one step of the summation. The pulse is based on the speed of the floating point multiplier. A multiply-add pulse was considered, but this would not suffice for most of the preprocessors, where two multiplies are required. Since 5 pipeline stages are required, the time to fill the pipelined Chebyshev processor is (5 * 160 ns) or 830 ns.

The preprocessor must be capable of range reduction and scaling for all of the functions. The preprocessor must then have 12 steps. The time to fill the preprocessor is (12 * 80 ns) or 960 ns. The postprocessor consists of two steps: a multiply and an add. The time to fill this pipeline is 160 ns.

The total latency, or time to fill the pipeline, is 1950 ns. One the pipeline is full, a result will be available every 160 ns.

Another benefit of expressing the Chebyshev polynominals in terms of x is that an entire stage of the Chebyshev pipeline can fit on one chip. The polynominals in terms of $T_n(x)$ will not fit on a single chip, so interchip communications must be considered.

Unified Pipeline for Preprocessing

An alternate design of the preprocessor for the sine, cosine, exponential, and natural logarithm is possible because the range reduction for these functions is similar. The order in which the steps for the preprocessors was presented for each of the functions in Chapter III was rearranged to become the unified preprocessor. Only a change in constants to the

preprocessing hardware is required. The hardware layout is shown in Figure 4.1. The constants for the preprocessor pipeline are listed in Table 4.5.

Table 4.5. Constants for static preprocessor pipeline.

Stage	Sine/Cosine	Exponential	Logarithm
Multiply1	$1/\pi$	1	2
Extractor	Integer Fraction	Integer Fraction	Exponent Mantissa
Multiply2	2	1	0.5, 0.6 or 0.75
Subtraction	1	0	1
Multiply3	1	1	Ln 2
Adder	0	0	Ln 1, Ln 1.2 or Ln 1.5

The extractor is different for the natural logarithm function than for the other functions. For the natural logarithm, the exponent and mantissa can be read directly from the input. But for sine, cosine, and exponential, a floor function followed by a subtraction is required. The result from the floor function is the integer part. The integer is subtracted from the input into the extractor which results in the fractional part.

Vector Wave Equation Processor

The Chebyshev processor discussed above is not the same one that would be used for the VWE. Since this processor would only compute the sine and cosine, only four pipeline stages would be required. Also, the number of steps for the preprocessor would be reduced to 4 steps. And since the VWE requires both the sine and cosine of the same argument, an extra multiplier and adder could be added to each stage of the pipeline to compute these functions

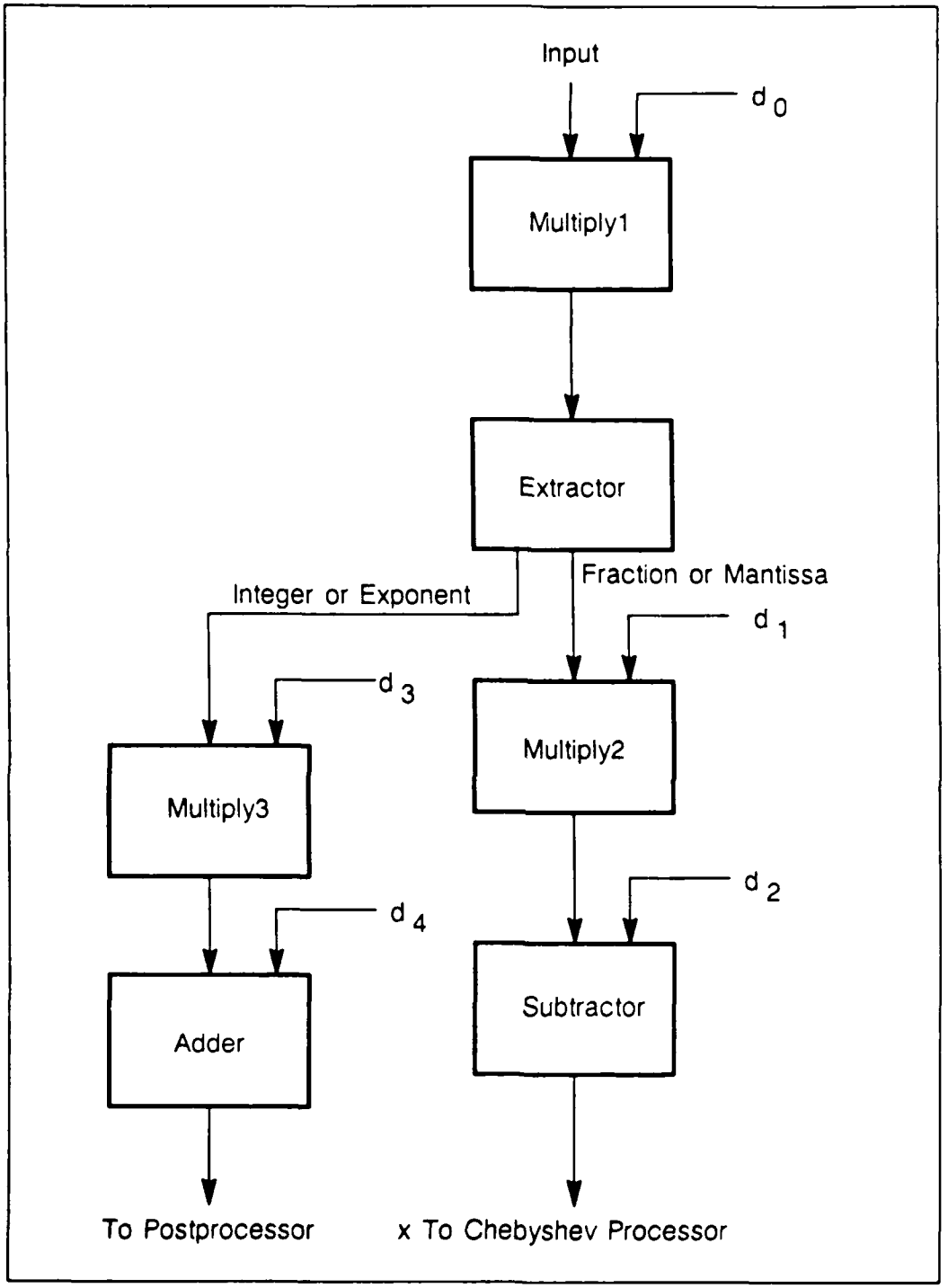


Figure 4.1. Unified Preprocessor Pipeline

simultaneously. The latency would be (4 * 160 ns) for the preprocessing, (4 * 160 ns for the Chebyshev hardware, and (2 * 80 ns) for the postprocessor. The total latency would be 1440 ns. After the pipeline is full, the sine and cosine of an argument would be output every 360 ns if the extra multiplier and adder are not added to each stage of the Chebyshev hardware. If they are added, then the sine and cosine would be output every 160 ns.

The CORDIC sine and cosine processor requires 24 iterations to produce the sine and cosine. Both the sine and cosine would be output every 2.2 us. The CORIC divider requires 48 iterations to produce the quotient. An output would be available every every 4.4 us.

The slowest computation in a pipeline limits the output of the entire processor. The 4.4 us time for the reciprocal could be reduced by pipelining within the CORDIC unit. Even without additional pipelining, the total time, once the pipeline is filled, for 720 observation points, each consisting of a dipole broken into 500 subelements, would require (4.4 us * 500 * 720) equals 1.57 seconds.

V. Conclusions and Recommendations

Conclusions

This study was motivated first by finding hardware to implement the elementary functions needed to compute the VWE. The end result is to be an interactive CAD/CAM sytem for computation of the electric and magnetic fields as well as the vector potential. An added motivation was the discovery of the Chebyshev polynominals and the fact that they could be transformed into a summation as a function of x, resulting in a pipelined hardware with one less element of hardware per stage.

Chapter II discusses the various algorithms for the computation of sine, cosine, and division. Both the CORDIC and Chebyshev algorithms were presented for the sine and cosine functions. The range normalization and scaling of the input arguments was presented. A division algorithm using the CORDIC equations was also presented. The hardware for each of the algorithms was also presented.

In Chapter III, the Chebyshev polynominals for elementary functions were presented. These functions include the sine, cosine, tangent, arctangent, exponential, and natural logarithm. Reduction techniques based on the properties of the functions were developed. The manipulation of the terms of the Chebyshev polynominals was also presented. The hardware for the preprocessing as well as the Chebyshev equations as a function of x were presented.

The advantages of transforming the Chebyshev polynomials as a function of x are threefold. First, the hardware is reduced for each stage of the Chebyshev processor pipeline. Specifically, one adder per stage is deleted. Second, the number of stages in the Cheybshev processor is reduced if only the odd or even $T_n(x)$'s are needed, as with the Chebyshev polynomial for cotangent. When using the $T_n(x)$ Chebyshev polynomials, each $T_n(x)$ must be

calculated, even if only some of them are necessary. But, in terms of x, the pipeline is set up to calculate either the even powers of x or increasing powers of x. And third, hardware is reduced in the preprocessor if the Chebyshev polynomial equation is not simply $T_n(x)$. For example, the sine Chebyshev polynomial equation would require two additional multipliers and one additional adder in the preprocessor.

And finally, in Chapter IV, state of the art hardware units for the basic hardware required by the CORDIC and Chebyshev algorithms was presented. Using this hardware, estimates as to the latency of the Chebyshev processor as well as the VWE processor was calculated. An alternate unified preprocessor pipeline was presented for calculating sine, cosine, exponential, and natural logarithm.

Recommendations

Further work is needed to find hardware to compute the square root for the VWE processor. An alternate hardware for computing the reciprocal is also needed as the CORDIC processor is not very fast compared to the computation times that are possible with the Chebyshev processor for sine and cosine.

Once the square root and division hardware is determined, hardware implementation of the VWE processor should be done. Issues of concern are the time-space tradeoff analysis, optimal scheduling and utilization, and VHSIC technology concerns. The algorithms should be modeled in the VHSIC Hardware Description Language (VHDL) to verify the accuracy of the algorithms and the hardware. The architecture may lend itself to wafer scale integration. The same issues are valid for the Chebyshev processor. Further work is needed to determine

the control section for the Chebyshev processor. The Chebyshev processor should also be modeled in VHDL.

Futher work is needed for the tangent and arctangent functions such that a division is not required. Equations for the Chebyshev polynomials can be evaluated such that the division may not be necessary. The unified preprocessor pipeline should accommodate these functions easily. Work in this area is mathematically intensive as the derivation of the constants, a_k, is difficult.

Research is also needed to determine if a Chebyshev polynomial can be found for 2^{-x} as discussed in Chapter III. The equation presented in Chapter III was not accurate enough for single precision. But if an equation can be found, the hardware required to compute e^x would be significantly reduced as this equation would not require 176 entries in a ROM table. The number of constants, b_k, would be halved for computing e^x as separate equations are now required for a positive and negative exponential. The equation using 2^x can solve for both a positive and negative exponential.

Appendix

Listing of Computer Programs

The following programs were written in Instant Pascal and were run on the Apple IIGS. All of the programs are interactive and compare the value as computed by the software to those obtained from either the CORDIC or Chebyshev algorithms.

Cordic Sine and Cosine

```
PROGRAM cordicsincos;
{
This program uses the CORDIC algorithm to compute the sine and cosine
of an input. The program performs range reduction, so that any value
can be input. The input is in degrees as the program converts the
input to radians. The program also computes the sine and cosine by
calling the predefined Instant Pascal functions which are computed
with extended precision (80 bits).
}
VAR
 x, y, z  : extended;
 compsin, compcos : extended;
 done :  boolean;
 sign, a, nodd  : integer;

PROCEDURE getvalue;
{
This procedure queries the operator for an input value in degrees and
converts degrees to radians.
}
BEGIN
 writeln('Input value for sin/cos in degrees. -109 to end');
 write('> ');
 readln(z);
 IF (z = -109) then
  done := true;
 z := z * (pi / 180);
END;

PROCEDURE computesincos;
{
This procedure calls the predefined sine and cosine functions and
computes the sine and cosine of the input
}
BEGIN
 compsin := sin(z);
 compcos := cos(z);
```

```
END;

PROCEDURE reduce;
{
This procedure reduces the input to the required range for the CORDIC
algorithm
}
VAR
 u : extended;
 n, signarg  :  integer;

BEGIN
 signarg := 1;
 IF (z < 0) THEN
   signarg := -1;
 u := (z / pi) + (signarg * 0.5);
 n := trunc(u);
 nodd := n MOD 2;
 z := ((u - n) - (0.5 * signarg)) * pi;
END;

PROCEDURE compute;
{
This procedure uses the CORDIC algorithm to compute the sine and
cosine of the reduced argument.
}
VAR
 tempx, tempy, tempz : extended;

BEGIN
 x := 0.607252936;
 y := 0;
 twopower := 1.0;
 FOR a := 0 TO 24 DO
 BEGIN
   IF (z < 0) THEN
     sign := 1
   ELSE
     sign := -1;
   tempz := z - (sign * arctan(twopower));
   tempx := x - (sign * y * twopower);
   tempy := y + (sign * x * twopower);
   z := tempz;
   x := tempx;
   y := tempy;
   twopower := twopower * 0.5;
 END;

 IF (nodd = 1) THEN
```

```
 BEGIN
  x := -1 * x;
  y := -1 * y;
 END;
END;

PROCEDURE output;
{
This procedure prints the sine and cosine to the screen.
}
BEGIN
 writeln('cordcos/compcos  ','cordsin/compsin');
 writeln(x:2:8,'  ',y:2:8);
 writeln(compcos:2:8,'  ',compsin:2:8);
END;

BEGIN
 done := false;
 WHILE (NOT done) DO
 BEGIN
  getvalue;
  computesincos;
  reduce;
  compute;
  IF (NOT done) THEN
  output;
 END;
END.
```

Chebyshev Expansion for Sine and Cosine

```
PROGRAM chebsincos;
{
This program uses the Chebyshev algorithm as a function of x to
compute the sine and cosine of an input. The program performs range
reduction, so that any value can be input. The input is in degrees as
the program converts the input to radians. The program also computes
the sine and cosine by calling the predefined Instant Pascal functions
which are computed with extended precision (80 bits).
}
VAR
 compsin, chebsin, compcos, chebcos, x  : extended;
 done :  boolean;
 signsin, signcos  : integer;

PROCEDURE getvalue;
{
This procedure queries the user for values to find the
sine and cosine of. The input is in degrees.
}
BEGIN
 writeln('Input value for sin/cos in degrees. -109 to end');
 write('> ');
 readln(x);
 IF (x = -109) then
  done := true;
 x := x * (pi / 180);
END;

PROCEDURE computesincos;
{
This procedure computes the sine and cosine using the predefined
functions found in the language
}
BEGIN
 compsin := sin(x);
 compcos := cos(x);
END;

PROCEDURE reduce;
{
This procedure reduces the input to the range required by the
Chebyshev processor. The sign of the output is also determined.
}
VAR
 u, fract :  extended;
 n, y  :  integer;
BEGIN
```

```
 u := x / pi;
 n := trunc(u);
 fract := u - n;
 x := (2 * fract);
 y := n MOD 2;
 signsin := 1;
 signcos := 1;
 IF ((y = 0) AND (x > 1.0) THEN
   signcos := -1;
 IF ((y = 1) AND (x <= 1.0) THEN
   signcos := -1;
 IF (y = 1) THEN
   signsin := -1;
END;

PROCEDURE compute;
{
This procedure uses the Chebyshev polynomials to compute the
sine and cosine of the reduced argument.
}
VAR
 answer1, answer2 : extended;

 PROCEDURE computechebcos;
 VAR
   a0, a2, a4, a6, a8 : extended;
   x2, x4, x6, x8 : extended;
 BEGIN
   a0 := 0.99999995327;
   a2 := -1.23369820792;
   a4 := 0.25365071056;
   a6 := -0.02081057280;
   a8 := 0.00085816320;
   x2 := x * x;
   x4 := x2 * x2;
   x6 := x4 * x2;
   x8 := x6 * x2;
   answer1 := a0 + (a2*x2)+(a4*x4)+(a6*x6)+(a8*x8);
 END;

 PROCEDURE computechebsin;
 VAR
   b0, b2, b4, b6, b8 : extended;
   x2, x4, x6, x8 : extended;
 BEGIN
   b0 := 1.57079628998;
   b2 := -0.64596335960;
   b4 := 0.07968847968;
   b6 := -0.00467222656;
```

```
  b8 := 0.00015081984;
  answer2 := x* (b0 + (b2*x2)+(b4*x4)+(b6*x6)+(b8*x8));
 END;

BEGIN
 IF (x > 1.0) THEN
 BEGIN
  x := x - 1;
  computechebcos;
  chebsin := answer1 * signsin;
  computechebsin;
  chebcos := answer2 * signcos;
 END
 ELSE
  computechebcos;
  chebsin := answer2 * signsin;
  computechebsin;
  chebcos := answer1 * signcos;
 END;
END;

PROCEDURE output;
{
This procedure prints the output to the screen.
}
BEGIN
 writeln('chebcos/compcos  ','chebsin/compsin');
 writeln(chebcos:2:8,'  ',chebsin:2:8);
 writeln(compcos:2:8,'  ',compsin:2:8);
END;

BEGIN
{
Main program
}
 done := false;
 WHILE (NOT done) DO
 BEGIN
  getvalue;
  computesincos;
  reduce;
  compute;
  IF (NOT done) THEN
  output;
 END;
END.
```

Chebyshev Expansion for Tangent

```
PROGRAM chebtangent;
{
This program uses the Chebyshev algorithm as a function of x to
compute the tangent of an input. The program performs range
reduction, so that any value can be input. The input is in degrees as
the program converts the input to radians. The program also computes
the tangent by calling the predefined Instant Pascal functions which
are computed with extended precision (80 bits). Since no function is
available to compute the tangent directly, both the sine and cosine
are computed, and the tangent is equal to sine/cosine.
}
VAR
 comptan, chebtan, compcos, compsin : extended;
 x, answertan, answercot : extended;
 done :  boolean;
 sign : integer;

PROCEDURE getvalue;
{
This procedure queries the user for an input in degrees.
Conversion to radians is done in this procedure.
}
BEGIN
 writeln('Input value for tangent in degrees. -109 to end');
 write('> ');
 readln(x);
 IF (x = -109) then
  done := true;
 x := x * (pi / 180);
END;

PROCEDURE computetan;
{
The predefined sine and cosine functions are called to compute
the tangent of the input.
}
BEGIN
 compsin := sin(x);
 compcos := cos(x);
 comptan := compsin / compcos;
END;

PROCEDURE reduce;
{
Range reduction is performed in this procedure, including the
sign of the final answer.
```

```
}
VAR
 u, fract : extended;
 n, y : integer;
BEGIN
 u := x / pi;
 n := trunc(u);
 fract := u - n;
 sign := 1;
 IF (y < 0) THEN
  sign := -1;
 x := fract * sign;
END;

PROCEDURE compute;
{
This procedure contains the procedures to compute the Chebyshev
tangent and cotangent functions. If the reduced argument is less than
(pi / 4), the tangent function is called. Otherwise, the cotangent
function is called.
}

 PROCEDURE computechebtan;
 VAR
  a1, a3, a5, a7, a9, a11 : extended;
  x3, x5, x7, x9, x11 : extended;
 BEGIN
  a1 := 0.78539686786;
  a3 := 0.16152638116;
  a5 := 0.03957327280;
  a7 := 0.01083740608;
  a9 := 0.00112678144;
  a11 := 0.00153917440;
  x3 := x * x * x;
  x5 := x3 * x * x;
  x7 := x5 * x * x;
  x9 := x7 * x * x;
  x11 := x9 * x * x;
  answertan := (a1*x)+(a3*x3)+(a5*x5)+(a7*x7)+(a9*x9)+(a11*x11);
 END;

 PROCEDURE computechebcot;
 VAR
  b0, b2, b4, b6, b8, b10 : extended;
  x2, x4, x6, x8, x10 : extended;
 BEGIN
  b0 := 1.27321935363;
  b2 := -0.26143595267;
  b4 := -0.01173517139;
```

```
  b6 := 0.00001332237;
  b8 := -0.00003841664;
  b10 := -0.00000294400;
  answercot := b0 + (b2*x2)+(b4*x4)+(b6*x6)+(b8*x8)+(b10*x10);
  answercot := answercot * (1/x);
 END;

BEGIN
{
Main body of compute
}
 IF ((x <= 0.25) OR (x >= 0.75)) THEN
 BEGIN
  IF (x >= 0.75) THEN
  BEGIN
   x := 1 - x;
   sign := -1 * sign;
  END;
 x := x * 4;
 computechebtan;
 chebtan := answertan * sign;
 END

 ELSE
 BEGIN
  IF (x <= 0.5) THEN
   x := 0.5 - x
  ELSE
   BEGIN
   sign := sign * -1;
   x := x - 0.5;
   END
  x := x * 4;
  computechebcot;
  chebtan := answercot * signsin;
 END;
END;

PROCEDURE output;
{
This procedure prints the output to the screen.
}
BEGIN
 writeln('chebtan   ','comptan');
 writeln(chebtan:2:8,'  ',comptan:2:8);
END;

{
Main procedure
```

```
}
BEGIN
 done := false;
 WHILE (NOT done) DO
 BEGIN
  getvalue;
  computetan;
  reduce;
  compute;
  IF (NOT done) THEN
  output;
 END;
END.
```

Chebyshev Expansion for Exponential

```
PROGRAM chebexp;
{
This program uses the Chebyshev algorithm as a function of x to
compute the exponential of an input. The program performs range
reduction, so that any value can be input. The program also computes
the exponential by calling the predefined Instant Pascal functions
which are computed with extended precision (80 bits).
}
VAR
 compexp, chebexp, x  : extended;
 done :  boolean;
 n  : integer;

PROCEDURE getvalue;
{
This procedure queries the user for an input.
}
BEGIN
 writeln('Input value for exponential. -109 to end');
 write('> ');
 readln(x);
 IF (x = -109) then
   done := true;
END;

PROCEDURE computeexp;
{
This procedure uses the predefined Pascal function to
compute the exponential of the input.
}
BEGIN
 compexp := exp(x);
END;

PROCEDURE reduce;
{
This procedure reduces the range of the input to that
acceptable for the Chebyshev polynomial equations.
}
BEGIN
 n := trunc(x);
 x := x - n;
END;

PROCEDURE compute;
{
```

```
This procedure consists of two subprocedures: one computes the
exponential of a positive input, the other computes the exponential
of a negative input. The main procedure multiplies the output of
the subprocedures by the exponential of the integer portion of the
reduce procedure. The exponential of the interger is performed by
the predefined function.
}
VAR
 answer : extended;

 PROCEDURE computeposexp;
 VAR
  a0, a2, a4, a6, a8 : extended;
  x2, x4, x6, x8 : extended;
 BEGIN
  a0 := 0.99999995327;
  a2 := -1.23369820792;
  a4 := 0.25365071056;
  a6 := -0.02081057280;
  a8 := 0.00085816320;
  x2 := x * x;
  x4 := x2 * x2;
  x6 := x4 * x2;
  x8 := x6 * x2;
  answer := a0 + (a2*x2)+(a4*x4)+(a6*x6)+(a8*x8);
 END;

 PROCEDURE computenegexp;
 VAR
  b0, b2, b4, b6, b8 : extended;
  x2, x4, x6, x8 : extended;
 BEGIN
  b0 := 1.57079628998;
  b2 := -0.64596335960;
  b4 := 0.07968847968;
  b6 := -0.00467222656;
  b8 := 0.00015081984;
  answer :=  (b0 + (b2*x2)+(b4*x4)+(b6*x6)+(b8*x8));
 END;

BEGIN
{
Main body of compute.
}
 IF (x > 0.0) THEN
 BEGIN
  computeposexp;
  chebexp := answer1 * exp(n);
 END
```

```
 ELSE
  BEGIN
  computenegexp;
  chebexp := answer2 * exp(n);
 END;
END;

PROCEDURE output;
{
This procedure prints the output to the screen.
}
BEGIN
 writeln('chebexp   ','compexp');
 writeln(chebexp:2:8, compexp:2:8);
END;

BEGIN
{
Main procedure
}
 done := false;
 WHILE (NOT done) DO
 BEGIN
  getvalue;
  computeexp;
  reduce;
  compute;
  IF (NOT done) THEN
  output;
 END;
END.
```

Chebyshev Expansion for Arctangent

```
PROGRAM chebartan;
{
This program uses the Chebyshev algorithm as a function of x to
compute the arctangent of an input. The program performs range
reduction, so that any value can be input. The program also computes
the arctangent by calling the predefined Instant Pascal functions
which are computed with extended precision (80 bits).
}
VAR
 compatn, chebatn, x  : extended;
 done, toobig :  boolean;
 n  : integer;

PROCEDURE getvalue;
{
This procedure queries the user for an input.
}
BEGIN
 writeln('Input value for arctangent. -109 to end');
 write('> ');
 readln(x);
 IF (x = -109) then
   done := true;
END;

PROCEDURE computeatn;
{
This procedure computes the arctangent of the input using
the predefined Pascal function for arctangent.
}
BEGIN
 compatn := arctan(x);
END;

PROCEDURE reduce;
{
This procedure reduces the input to a value that meets the
input requirements of the Chebyshev polynomial equation.
If the input is greater that 1, the reciprical of the input
is calculated.
}
BEGIN
 toobig := FALSE;
 IF (x > 1) THEN
 BEGIN
   toobig := TRUE;
```

```
  x := 1 / x;
 END;
END;

PROCEDURE compute;
{
This procedure computes the arctangent using the Chebyshev
polynomial equations.  If the unreduced input is greater
than 1, then the output from the Chebyshev equations must be
subtracted from (pi/2).
}
VAR
 answer : extended;

 PROCEDURE computeatn;
 VAR
  a0, a2, a4, a6, a8 : extended;
  x2, x4, x6, x8 : extended;
 BEGIN
  a0 := 0.99999990304;
  a2 := -0.33332184597;
  a4 := 0.19961556588;
  a6 := -0.13751623066;
  a8 := 0.07726269923;
  x2 := x * x;
  x4 := x2 * x2;
  x6 := x4 * x2;
  x8 := x6 * x2;
  answer := x*(a0 + (a2*x2)+(a4*x4)+(a6*x6)+(a8*x8));
 END;

BEGIN
{
Main body of compute.
}
  computeatn;
  IF (toobig) THEN
   chebatn := (pi / 2) - answer
  ELSE
   chebatn := answer;
END;

PROCEDURE output;
{
This procedure prints the output to the screen.
}
BEGIN
 writeln('chebatn   ','compatn');
 writeln(chebatn:2:8, compatn:2:8);
```

```
END;

BEGIN
{
Main body of program.
}
 done := false;
 WHILE (NOT done) DO
 BEGIN
  getvalue;
  computeatn;
  reduce;
  compute;
  IF (NOT done) THEN
  output;
 END;
END.
```

Chebyshev Expansion for Natural Logarithm

```
PROGRAM chebnatlog;
{
This program uses the Chebyshev algorithm as a function of x to
compute the natural logarithm of an input. The program does not
perform range reduction because access to individual bits is not
supported by this version of Pascal. Only values between 0.75 and 1.0
can be input. The program also computes the natural logarithm by
calling the predefined Instant Pascal functions which are computed
with extended precision (80 bits). But the output of this program
can be used to manually compute the natural logarithm of values
not in the required range.
}
VAR
 compnatlog, chebnatlog, x  : extended;
 done :  boolean;
 n  : integer;

PROCEDURE getvalue;
{
This procedure queries the user for input.
}
BEGIN
 writeln('Input value for natural log. -109 to end');
 write('> ');
 readln(x);
 IF (x = -109) then
  done := true;
 IF ((x < 0.75) OR (x > 1.0)) THEN
 BEGIN
 writeln('Input not in required range, 0.75 to 1.0.');
 write('> ');
 readln(x);
 END;
END;

PROCEDURE computenatlog;
{
This procedure uses the predefined function to compute
the natural logarithm of the input.
}
BEGIN
 compnatlog := ln(x);
END;

PROCEDURE reduce;
{
```

```
This procedure performs range reduction so that the
natural logarithm of the input is computed, not the
natural logarithm of 1 + the input.
}
BEGIN
 x := 1 - x;
END;

PROCEDURE compute;
{
This procedure uses the Chebyshev polynomial equations
to compute the natural logarithm of the reduced input.
}
VAR
 answer : extended;

 PROCEDURE computenatlog;
 VAR
  a0, a2, a4, a6, a8 : extended;
  x2, x4, x6, x8 : extended;
 BEGIN
  a0 := -0.00000000958;
  a1 := 0.99999686448;
  a2 := -0.50016647931;
  a3 := 0.33005325226;
  a4 := -0.28021885649;
  a5 := 0.06217418014;
  x2 := x * x;
  x3 := x2 * x;
  x4 := x3 * x;
  x5 := x4 * x;
  x6 := x5 * x;
  answer := a0 + (a1*x)+(a2*x2)+(a3*x3)+(a4*x4)+(a5*x5);
 END;

BEGIN
  computenatlog;
  chebnatlog := answer;
END;

PROCEDURE output;
{
This procedure outputs the answers to the screen.
}
BEGIN
 writeln('chebnatlog  ','compnatlog');
 writeln(chebnatlog:2:8, compnatlog:2:8);
END;
```

```
BEGIN
{
Body of main procedure.
}
 done := false;
 WHILE (NOT done) DO
 BEGIN
  getvalue;
  computenatlog;
  reduce;
  compute;
  IF (NOT done) THEN
  output;
 END;
END.
```

Bibliography

1. Balanis, Constantine A. *Antenna Theory*. New York: Harper & Row, Publishers, Inc., 1982.

2. Cosnard, M. and others. "The FELIN Arithmetic Coprocessor Chip," *Proceedings on the Eighth Symposium on Computer Arithmetic*. 107-112. Washington: IEEE, 1987.

3. Daggett, D. H. "Decimal-Binary Conversions in CORDIC," *IRE Transactions on Electronic Computers, EC-9:* 335-339 (September 1959).

4. Fandrianto, Jan. "Algorithm for High Speed Shared Radix 4 Division and Radix 4 Square-root," *Proceedings on the Eighth Symposium on Computer Arithmetic*. 73-79. Washington: IEEE, 1987.

5. Fox, L. and I. B. Parker. *Chebyshev Polynomials in Numerical Anaylsis*. London: Oxford University Press, 1968.

6. Gallagher, David. *Rapid Prototyping of Application Specific Processors*. MS Thesis, AFIT/GE/ENG/87D-19. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, December 1987.

7. Hoyt, 1st Lt Brian A. *Digital Algorithm Specification for the VLSI Implementation of the Electromagnetic Field of an Arbitray Current Source*. MS Thesis, AFIT/GE/ENG/86D-18. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, December 1986.

8. Hwang, Kai and Faye A. Briggs. *Computer Architecture and Parallel Processing*. New York: McGraw-Hill, Inc., 1984.

9. Hwang, Kai and others. "Evaluating Elementary Functions with Chebyshev Polynomials on Pipeline Nets," *Proceedings on the Eighth Symposium on Computer Arithmetic*. 121-128. Washington: IEEE, 1987.

10. *IEEE Standard 754 for Binary Floating-Point Arithmetic*. New York: IEEE Press, 1985.

11. Jones, Capt Lawrence E. *Algorithm Definition for the VLSI Design Implementation of the Electromagnetic Radiation Integral*. MS Thesis, AFIT/GE/ENG/85-23. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, December 1985.

12. Lyusternik, L. A. and others. *Handbook for Computing Elementary Functions*. Oxford: Pergamon Press, 1965.

13. Steer, D. G. and S. R. Penstone. "Digital Hardware for Sine-Cosine Function," *IEEE Transactions on Computers, C-26:* 1283-1286 (December 1977).

14. Strass, Capt Jack L. *Architectural Implications of a Parallel Computational Approach to the Vector Wave Equation*. MS Thesis, AFIT/GE/ENG/87M-5. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, May 1987.

15. Volder, Jack E. "The CORDIC Trigonometric Computing Technique," *IRE Transactions on Electronic Computers, EC-8:* 330-334 (August 1959).

Vita

Captain Mickey J. Bailey was born on September 2, 1956 in Dansville, New York. He graduated from Keshequa Central School in 1974 and attended Houghton College for two years prior to enlisting in the Air Force. As an enlisted member he performed duties in the 6453X career field (Inventory Management Specialist). He was accepted into the airman Education and Commissioning Program in 1980 and graduated from the University of South Carolina in 1982 with a BSEE. After receiving a commission through the Officiers Training School, he was assigned to the Strategic Communications Directorate at the Electronic Systems Division, Hanscom AFB, Massachusettes where he served as both the Chief Project Engineer and Deputy Program Manager for the Aircraft Alerting Communications EMP Upgrade Program. He entered the School of Engineering at the Air Force Institute of Technology, Wright-Patterson AFB, Ohio in May 1986.

Permanent Address: 9939 Oakland Street

Dalton, NY 14836

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GE/ENG/87D-3	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Engineering 6b. OFFICE SYMBOL (If applicable) AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB, OH 45433-6583	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11. TITLE (Include Security Classification)
HIGH SPEED TRANSCENDENTAL ELEMENTARY FUNCTION ARCHITECTURE IN SUPPORT OF THE VECTOR WAVE EQUATION (VWE)

12. PERSONAL AUTHOR(S)
Mickey J. Bailey, B.S., Captain, USAF

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
MS Thesis	FROM _____ TO _____	1987 December	91

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	
12	01		CORDIC Algorithm, Chebyshev Polynomials, Range Reduction, Pipelining
12	03		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Thesis Chairman: Joseph DeGroat, Major, USAF

IAW AFR 190-1
31 Dec 87
Professional Development
Air Force Institute of Technology (AU)
Wright-Patterson AFB OH 45433

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Joseph DeGroat, Major, USAF	22b. TELEPHONE (Include Area Code) (513) 255-5633 22c. OFFICE SYMBOL AFIT/ENG

DD Form 1473, JUN 86 *Previous editions are obsolete.*

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

In support of a Very High Speed Integrated Circuit (VHSIC) class processor for computation of a set of equations known as the Vector Wave Equations (VWE), certain elementary fuctions including sine, cosine, and division are required. These elementary functions are the bottlenecks in the VWE processor. Floating point multipliers and adders comprise the remainder of the pipeline stages in the VWE processor.

To speed up the computation of the elementary functions, pipelining within the functions is considered. To compute sine, cosine, and division, the CORDIC algorithm is presented. Another method for computation of sine and cosine is the expansion of the Chebyshev polynominals.

The equations for the CORDIC processor are recursive and the resulting hardware is very simple, consisting of three adders, three shifters, and lookup table for some of the coefficients. The shifters replace the multiplies, because in binary, i right shifts is the same as multiplying by 2^i.

The expansion of the Chebyshev polynominals can be used to compute other trigonometric functions as well as the exponential and logarithmic functions. The expansion of the Chebyshev polynominals can be used as a mathematic coprocessor. From these equations, a pipelined architecture can be realized that results in very fast computation times. The transformation of these equations as a function of x instead of the Chebyshev polynominals produces an architecture which requires less hardware, resulting in even faster computation times.

END

DATE

FILMED

APRIL

1988

DTIC